

Android™



COMO PROGRAMAR

SEGUNDA EDIÇÃO

com introdução a Java



Paul Deitel • Harvey Deitel • Abbey Deitel





D324a Deitel, Paul.

Android : como programar [recurso eletrônico] / Paul Deitel, Harvey Deitel, Abbey Deitel ; tradução: João Eduardo Nóbrega Tortello. – 2. ed. – Porto Alegre : Bookman, 2015.

Editado como livro impresso em 2015.
ISBN 978-85-8260-348-2

1. Programação - Android. I. Deitel, Harvey. II. Deitel, Abbey. III. Título.

CDU 004.438Android

Paul Deitel • Harvey Deitel • Abbey Deitel
Deitel & Associates, Inc.

Android™



COMO PROGRAMAR

SEGUNDA EDIÇÃO

Tradução:

João Eduardo Nóbrega Tortello

Versão impressa
desta obra: 2015



2015

Obra originalmente publicada sob o título
Android: How to Program, 2nd Edition
ISBN 0-13-376403-6 / 978-0-13-376403-1

Tradução autorizada a partir do original em língua inglesa da obra intitulada ANDROID HOW TO PROGRAM, 2ND EDITION, de autoria de PAUL DEITEL, HARVEY DEITEL, ABBEY DEITEL, publicada pela Pearson Education, Inc., sob o selo Prentice Hall, © 2015.

A edição em língua portuguesa desta obra é publicada por Bookman Companhia Editora Ltda., uma divisão do Grupo A Educação S.A., © 2015.

DEITEL, o inseto com os dois polegares para cima e DIVE-INTO são marcas registradas de Deitel & Associates, Inc. Java é uma marca registrada de Oracle e/ou suas afiliadas. Google, Android, Google Play, Google Maps, Google Wallet, Nexus, YouTube, AdSense e AdMob são marcas registradas de Google, Inc. Microsoft® e Windows® são marcas registradas de Microsoft Corporation nos Estados Unidos e/ou em outros países. Outros nomes podem ser marcas registradas de seus respectivos detentores.

Gerente editorial: *Arysinha Jacques Affonso*

Colaboraram nesta edição:

Editora: *Mariana Belloli*

Leitura final: *Susana de Azeredo Gonçalves*

Capa: *Kaêle Finalizando Ideias*, arte sobre capa original

Editoração: *Techbooks*

Reservados todos os direitos de publicação, em língua portuguesa, à
BOOKMAN EDITORA LTDA., uma empresa do GRUPO A EDUCAÇÃO S.A.
Av. Jerônimo de Ornelas, 670 – Santana
90040-340 – Porto Alegre – RS
Fone: (51) 3027-7000 Fax: (51) 3027-7070

É proibida a duplicação ou reprodução deste volume, no todo ou em parte, sob quaisquer formas ou por quaisquer meios (eletrônico, mecânico, gravação, fotocópia, distribuição na Web e outros), sem permissão expressa da Editora.

Unidade São Paulo
Av. Embaixador Macedo Soares, 10.735 – Pavilhão 5 – Cond. Espace Center
Vila Anastácio – 05095-035 – São Paulo – SP
Fone: (11) 3665-1100 Fax: (11) 3667-1333

SAC 0800 703-3444 – www.grupoa.com.br

IMPRESSO NO BRASIL
PRINTED IN BRAZIL

*Em memória de Amar G. Bose, professor do MIT e
fundador e presidente da Bose Corporation:*

*Foi um privilégio sermos seus alunos – e membros
da nova geração de Deitels que ouvia nosso pai dizer
como suas aulas o inspiraram a dar o melhor de si.*

*Você nos ensinou que, se correremos atrás dos problemas
realmente difíceis, coisas excelentes podem acontecer.*

*Harvey Deitel
Paul e Abbey Deitel*

Esta página foi deixada em branco intencionalmente.



Prefácio

Faça a melhor ratoeira e o mundo abrirá um caminho até sua porta.

Ralph Waldo Emerson

*A ciência, a tecnologia e as várias formas de arte
reúnem a humanidade em um sistema único e interligado.*

Zhores Aleksandrovich Medvede

Bem-vindo ao dinâmico mundo do desenvolvimento de aplicativos Android para smartphones e tablets com o SDK (Software Development Kit) para Android, a linguagem de programação Java™, o IDE (integrated development environment ou ambiente de desenvolvimento integrado) Development Tools para Android e ao novo e em franca evolução Android Studio. Apresentamos tecnologias de computação móvel de ponta para alunos, professores e desenvolvedores de software profissionais.

Android: Como Programar, 2ª edição

Com este livro sem igual – a segunda edição do primeiro livro-texto sobre ciência da computação com Android do mundo –, você pode aprender sobre Android sem conhecer Java e mesmo sendo iniciante em programação. Esta obra contém 300 páginas de uma introdução completa sobre os conceitos de programação básica com Java, que você vai precisar quando desenvolver aplicativos Android. O conteúdo Java é adequado aos iniciantes em programação.

Este livro foi feito combinando

- nosso livro profissional *Android para Programadores: Uma abordagem baseada em aplicativos, 2ª edição*
- conteúdo introdutório básico e condensado sobre programação orientada a objetos com Java, de nosso livro acadêmico *Java How to Program, 9/e*
- centenas de perguntas com respostas curtas sobre Android e exercícios de desenvolvimento de aplicativo criados para este livro.

Percorremos o material sobre Android, especialmente os aplicativos totalmente codificados nessa linguagem, e enumeramos os recursos em Java que serão necessários para a construção desses e de aplicativos semelhantes. Então, extraímos o conteúdo Java correspondente do livro *Java How to Program, 9/e*. Essa é uma obra de 1500 páginas, de modo que foi desafiador reduzir esse conteúdo enorme e mantê-lo amigável, mesmo para iniciantes em programação.

Quando estudar o conteúdo sobre Android, você vai pensar como desenvolvedor desde o começo. Vai estudar e construir muitos exemplos reais e vai se deparar com os tipos de desafios encontrados pelos desenvolvedores profissionais. Vamos remetê-lo à documentação online e a fóruns em que você poderá encontrar mais informações e obter respostas para suas perguntas. Vamos estimulá-lo a ler, modificar e melhorar código-fonte aberto, como parte de seu processo de aprendizagem.

Público-alvo

Este livro serve para várias pessoas. Mais comumente, será utilizado em cursos universitários e em cursos profissionalizantes e por pessoas familiarizadas com a programação orientada a objetos, mas que podem ou não conhecer Java e que querem aprender sobre desenvolvimento de aplicativos Android.

O livro também pode ser usado em cursos introdutórios, destinados a iniciantes em programação. Recomendamos às escolas que normalmente oferecem muitos cursos básicos sobre Java que pensem na possibilidade de destinar um ou dois módulos do curso para os alunos mais avançados que tenham pelo menos certa experiência anterior em programação e que queiram estudar com afinco para aprender bastante sobre Java e Android no prazo de um semestre. Talvez as escolas queiram designar esses módulos como “avançados”. O livro funciona particularmente bem em seqüências de programação introdutória de dois semestres, em que a introdução a Java é abordada primeiro.

Cursos de desenvolvimento de aplicativos

Em 2007, a Stanford ofereceu um novo curso, chamado Creating Engaging Facebook Apps (Criando aplicativos atraentes para o Facebook). Os alunos trabalhavam em equipes desenvolvendo aplicativos, alguns dos quais acabaram entre os 10 principais aplicativos do Facebook, rendendo milhões de dólares aos alunos desenvolvedores.¹ Esse curso obteve amplo reconhecimento por estimular a criatividade e o trabalho em equipe dos alunos. Atualmente, várias faculdades oferecem cursos de desenvolvimento de aplicativos para diferentes redes sociais e plataformas móveis, como Android e iOS. Incentivamos você a ler o plano de estudos de desenvolvimento de aplicativos móveis online e assistir aos vídeos do YouTube™ criados por instrutores e alunos de muitos desses cursos.

Ecossistema Android: concorrência, inovação, crescimento explosivo e oportunidades

As vendas de dispositivos Android e os downloads de aplicativos estão crescendo exponencialmente. A primeira geração de telefones Android foi lançada em outubro de 2008. Um estudo da Strategy Analytics mostrou que, em outubro de 2013, o Android tinha 81,3% da fatia de mercado global de smartphones, comparados com os 13,4% da Apple, 4,1% da Microsoft e 1% do Blackberry.² De acordo com um relatório do IDC, no final do primeiro trimestre de 2013, o Android tinha 56,5% de participação no mercado global de tablets, comparados com os 39,6% do iPad da Apple e 3,7% dos tablets Microsoft Windows.³

Atualmente existem mais de 1 bilhão de smartphones e tablets Android em uso,⁴ e mais de 1,5 milhão de aparelhos Android sendo ativados diariamente.⁵ De acordo com o IDC, a Samsung é líder na fabricação de Android, contando com quase 40% das entregas de aparelhos Android no terceiro trimestre de 2013.

¹ <http://www.businessinsider.com/these-stanford-students-made-millions-taking-a-class-on-facebook-2011-5>.

² <http://blogs.strategyanalytics.com/WSS/post/2013/10/31/Android-Captures-Record-81-Percent-Share-of-Global-Smartphone-Shipments-in-Q3-2013.aspx>.

³ <http://www.idc.com/getdoc.jsp?containerId=prUS24093213>.

⁴ <http://www.android.com/kitkat>.

⁵ <http://www.technobuffalo.com/2013/04/16/google-daily-android-activations-1-5-million>.

Bilhões de aplicativos foram baixados do Google Play™ – a loja do Google para aplicativos Android. As oportunidades para os desenvolvedores de aplicativos Android são imensas. A acirrada competição entre as plataformas móveis populares e entre as empresas de telefonia móvel está levando à rápida inovação e à queda nos preços. A concorrência entre as dezenas de fabricantes de dispositivos Android impulsiona a inovação de hardware e software dentro da comunidade Android.

Abordagem baseada em aplicativos

Este livro se baseia em nossa *abordagem baseada em aplicativos* – apresentamos os conceitos no contexto de *sete aplicativos Android funcionais completos*. Iniciamos cada capítulo com uma *introdução* ao aplicativo em questão, um *teste* mostrando um ou mais *exemplos de execução* e uma *visão geral das tecnologias*. Construímos a interface gráfica do usuário e os arquivos de recurso do aplicativo. Em seguida, passamos a *acompanhar detalhadamente o código-fonte* do aplicativo, discutindo os conceitos de programação e demonstrando a funcionalidade das APIs do Android utilizadas. Todo o código-fonte está disponível no endereço www.grupoa.com.br. Recomendamos abrir o código-fonte no IDE à medida que você ler o livro. A Figura 1 lista os aplicativos do livro e as principais tecnologias utilizadas para construir cada um deles.

Aplicativo	Tecnologias
Capítulo 2, aplicativo Welcome	O Android Developer Tools (o IDE do Eclipse e o ADT Plugin), projeto visual de interface gráfica do usuário, layouts, componentes TextView e ImageView, acessibilidade e internacionalização.
Capítulo 3, aplicativo Tip Calculator	Componentes GridLayout, LinearLayout, EditText, SeekBar, tratamento de eventos, NumberFormat e definição de funcionalidade de aplicativo com Java.
Capítulo 4, aplicativo Twitter® Searches	SharedPreferences, coleções, ImageButton, ListView, ListActivity, ArrayAdapter, objetos Intent implícitos e componentes AlertDialog.
Capítulo 5, aplicativo Flag Quiz	Fragmentos, menus, preferências, AssetManager, animações com tween, componentes Handler e Toast, objetos Intent explícitos, layouts para várias orientações de dispositivo.
Capítulo 6, aplicativo Cannon Game	Deteção de toques, animação quadro a quadro, elementos gráficos, som, threads, SurfaceView e SurfaceHolder.
Capítulo 7, aplicativo Doodlz	Elementos gráficos bidimensionais, Canvas, Bitmap, acelerômetro, SensorManager, eventos multitouch, MediaStore, impressão e modo imersivo.
Capítulo 8, aplicativo Address Book	Componentes AdapterView e Adapter.

Figura 1 Aplicativos presentes no livro impresso.

Capítulos online e atualizações do livro

Este livro também tem um site (em inglês) que contém capítulos adicionais sobre desenvolvimento de aplicativos; eles apresentam animação de propriedades, serviços de jogos do Google Play, vídeo, síntese e reconhecimento de fala, GPS, a API Maps, a bússola, serialização de objetos, aplicativos capacitados para a Internet, gravação e reprodução

de áudio, Bluetooth®, aplicativos móveis HTML5 e muito mais. **Para saber sobre os capítulos online e obter atualizações contínuas do livro, visite**

<http://www.deitel.com/books/AndroidHTP2>

Entre nas comunidades da Deitel no Facebook® (<http://www.deitel.com/deitelfan>), Twitter® (@deitel), LinkedIn® (<http://bit.ly/DeitelLinkedIn>), Google+™ (<http://google.com/+DeitelFan>) e YouTube™ (<http://youtube.com/user/DeitelTV>), e assine a newsletter *Deitel® Buzz Online* (<http://www.deitel.com/newsletter/subscribe.html>).

Aviso sobre direitos de cópia e licença de código

Todos os códigos e aplicativos Android do livro são protegidos por direitos de cópia pela Deitel & Associates, Inc. Os exemplos de aplicativos Android são licenciados sob a Creative Commons Attribution 3.0 Unported License (<http://creativecommons.org/licenses/by/3.0/>) e não podem ser reutilizados de forma alguma em aulas e livros acadêmicos, seja no formato impresso ou digital. Além disso, os autores e a editora não dão qualquer garantia, expressa ou implícita, com relação a esses programas ou à documentação contida neste livro. Os autores e a editora não se responsabilizam por quaisquer danos, casuais ou consequentes, ligados ou provenientes do fornecimento, desempenho ou uso desses programas. Você está livre para usar os aplicativos do livro como base para seus próprios aplicativos, ampliando a funcionalidade existente. Caso tenha quaisquer dúvidas, entre em contato conosco pelo endereço deitel@deitel.com.

Atualize-se sobre Java e XML

A parte deste livro relativa ao Android presume que você já conhece Java e programação orientada a objetos. Se esse não for o caso, os apêndices fornecem uma introdução condensada e amigável sobre Java e técnicas de programação orientada a objetos, necessárias para desenvolver aplicativos Android. Caso esteja interessado em aprender Java com mais profundidade, talvez você queira examinar a ampla abordagem em nosso livro-texto *Java How to Program, 10/e* (<http://www.deitel.com/books/jhtp10>).

Grças às ferramentas de desenvolvimento para Android aprimoradas, pudemos eliminar quase toda marcação XML nesta edição. Ainda existem dois pequenos arquivos XML fáceis de entender que precisam ser manipulados. Caso não conheça XML, consulte os seguintes tutoriais online (em inglês):

- http://www.deitel.com/articles/xml_tutorials/20060401/XMLBasics/
- http://www.deitel.com/articles/xml_tutorials/20060401/XMLStructuringData/
- <http://www.ibm.com/developerworks/xml/newto/>
- http://www.w3schools.com/xml/xml_what_is.asp

Principais recursos deste livro

- **Android SDK 4.3 e 4.4.** Abordamos vários recursos novos do SDK (Software Development Kit) 4.3 e 4.4 para Android. [*Obs.:* Os aplicativos deste livro são confi-

gurados para serem executados em dispositivos Android com Android 4.3 e superiores; contudo, a maioria funcionará no 4.0 e superiores, trocando seu SDK mínimo exigido.]

- **Fragments.** A partir do Capítulo 5, utilizamos Fragments para criar e gerenciar partes da interface gráfica de cada aplicativo. Você pode combinar vários fragmentos para criar interfaces do usuário que tiram proveito dos tamanhos de tela dos tablets. Pode também trocar facilmente os fragmentos para tornar suas interfaces gráficas mais dinâmicas, como fará no Capítulo 8.
- **Suporte para vários tamanhos e resoluções de tela.** Ao longo dos capítulos com aplicativos, demonstramos como utilizar os mecanismos do Android para escolher recursos automaticamente (layouts, imagens, etc.), com base no tamanho e na orientação do dispositivo.
- **Abordagem do ADT (Android Development Tools) baseado no Eclipse no livro impresso.** O ambiente de desenvolvimento integrado (IDE) gratuito ADT (Android Development Tools) – o qual inclui o Eclipse e o plugin ADT –, combinado com o JDK (Java Development Kit), também gratuito, fornecem todo o software necessário para criar, executar e depurar aplicativos Android, exportá-los para distribuição (por exemplo, carregá-los no Google Play™) e muito mais.
- **Android Studio.** É o IDE preferido para o futuro desenvolvimento de aplicativos Android. Como esse IDE está evoluindo rapidamente, nossas discussões sobre ele se encontram online, no endereço:

<http://www.deitel.com/books/AndroidHTP2>

- **Modo imersivo.** A barra de status na parte superior da tela e os botões de menu na parte inferior podem ser ocultos, permitindo que seus aplicativos ocupem uma parte maior da tela. Os usuários podem acessar a barra de status fazendo um pressionamento forte (swipe) de cima para baixo na tela, e a barra de sistema (com os botões voltar, iniciar e aplicativos recentes) fazendo um pressionamento forte de baixo para cima.
- **Framework de impressão.** O Android 4.4 KitKat permite adicionar aos seus aplicativos funcionalidade de impressão, como localizar impressoras disponíveis via Wi-Fi ou na nuvem, selecionar o tamanho do papel e especificar as páginas a serem impressas.
- **Teste em smartphones, em tablets e no emulador Android.** Para obter a melhor experiência no desenvolvimento de aplicativos, você deve testar seus aplicativos em smartphones e tablets Android reais. Você ainda pode ter uma experiência significativa usando o emulador Android (consulte a seção “Antes de começar”); contudo, ele utiliza muito poder de processamento e pode ser lento – particularmente em jogos que têm muitas peças móveis. No Capítulo 1, mencionamos alguns recursos do Android que não são suportados no emulador.
- **Multimídia.** Os aplicativos do livro impresso utilizam uma ampla variedade de recursos multimídia para Android, incluindo elementos gráficos, imagens, animação quadro a quadro e áudio. Os aplicativos dos capítulos online utilizam animação de propriedades, vídeo, síntese e reconhecimento de fala.

- **Boas práticas para Android.** Obedecemos às boas práticas aceitas para Android, assinalando-as nos acompanhamentos de código detalhados. Para obter mais informações, visite <http://developer.android.com/guide/practices/index.html>.
- **O conteúdo em Java dos apêndices pode ser usado com Java SE 6 ou superior.**
- **Tratamento de exceções em Java.** Integramos o tratamento de exceções básico no início do conteúdo em Java e, então, apresentamos um tratamento mais detalhado no Apêndice H; utilizamos tratamento de exceções em todos os capítulos sobre Android.
- **Classes Arrays e ArrayList; coleções.** O Apêndice E aborda a classe Arrays – a qual contém métodos para realizar manipulações comuns de array – e a classe genérica ArrayList – a qual implementa uma estrutura de dados do tipo array que pode ser dimensionada dinamicamente. O Apêndice J apresenta as coleções genéricas do Java que são utilizadas frequentemente em nossa abordagem sobre Android.
- **Multithread Java.** Manter a rapidez de resposta do aplicativo é fundamental para a construção de aplicativos Android robustos e exige uso extensivo de multithread Android. O Apêndice J apresenta os fundamentos do multithread para que você possa entender como usamos a classe AsyncTask do Android no Capítulo 8.
- **Apresentação da interface gráfica do usuário.** O Apêndice I apresenta o desenvolvimento de interface gráfica do usuário em Java. O Android fornece seus próprios componentes de interface gráfica do usuário, de modo que esse apêndice apresenta alguns componentes de interface gráfica do usuário em Java e se concentra nas classes aninhadas e nas classes internas anônimas, as quais são usadas extensivamente para tratamento de evento nas interfaces gráficas do usuário do Android.

Trabalho com aplicativos de código aberto

Existem online inúmeros aplicativos Android de código-fonte aberto gratuitos, os quais são recursos excelentes para se aprender desenvolvimento de aplicativos Android. Incentivamos o download de aplicativos de código aberto e a leitura do código para entender como ele funciona. Em todo o livro, você vai encontrar exercícios de programação que pedem para modificar ou melhorar os aplicativos de código-fonte aberto existentes. Nosso objetivo é expô-lo a problemas interessantes que também poderão inspirá-lo a criar novos aplicativos usando as mesmas tecnologias. **Aviso: os termos das licenças de código aberto variam consideravelmente.** Alguns permitem usar o código-fonte do aplicativo livremente para qualquer propósito, enquanto outros estipulam que o código está disponível apenas para uso pessoal – não para a criação de aplicativos para venda ou publicamente disponíveis. **Leia atentamente as concessões do licenciamento. Se quiser criar um aplicativo comercial baseado em um aplicativo de código aberto, você deve pensar na possibilidade de conseguir que um advogado com experiência em propriedade intelectual leia a licença – esses advogados cobram caro.**

Recursos pedagógicos

Sintaxe sombreada. Por questões de clareza, sombreamos a sintaxe do código de modo similar ao uso de cores na sintaxe do Eclipse e do Android Studio. Nossas convenções são as seguintes:

os comentários aparecem em cinza
os valores constantes e literais aparecem em cinza e negrito
as palavras-chave aparecem em preto e negrito
o restante do código aparece em preto

Realce de código. Destacamos os principais segmentos de código de cada programa com retângulos em cinza-claro.

Uso de fontes para dar ênfase. Utilizamos várias convenções de fonte:

- As ocorrências explicativas de termos importantes aparecem em **negrito** para fácil referência.
- Os componentes do IDE na tela aparecem na fonte **Helvetica em negrito** (por exemplo, o menu **File**).
- Código-fonte de programa aparece na fonte **Lucida** (por exemplo, `int x = 5;`).

Neste livro, você vai criar interfaces gráficas do usuário utilizando uma combinação de programação visual (apontar e clicar, arrastar e soltar) e escrita de código.

Utilizamos fontes diferentes ao nos referirmos aos elementos de interface gráfica do usuário em código de programa e aos elementos de interface gráfica do usuário exibidos no IDE:

- Componentes de interface gráfica do usuário que criamos em um programa aparecem com seu nome de classe e nome de objeto na fonte **Lucida** – por exemplo, “`Button saveContactButton`”.
- Componentes de interface gráfica do usuário que fazem parte do IDE aparecem com o texto do componente na fonte **Helvetica em negrito** e com fonte de texto normal para o tipo do componente – por exemplo, “o menu **File** ou o botão **Run**”.

Uso do caractere >. Usamos o caractere > para indicar a seleção de um item em um menu. Por exemplo, usamos a notação **File > New** para indicar que você deve selecionar o item **New** no menu **File**.

Código-fonte. Todo o código-fonte do livro está disponível para download no endereço:

www.grupoa.com.br

Cadastre-se gratuitamente, encontre e acesse a página do livro por meio do campo de busca e clique no link **Conteúdo Online** para fazer download dos códigos.

Objetivos do capítulo. Cada capítulo começa com uma lista de objetivos de aprendizado.

Figuras. Existem centenas de tabelas, listagens de código-fonte e capturas de tela.

Engenharia de software. Enfatizamos a clareza e o desempenho dos programas, e nos concentramos na construção de software bem projetado e orientado a objetos.

Exercícios de revisão e respostas. Foram incluídos exercícios de revisão e respostas para autoestudo.

Exercícios com um toque de atualidade. Trabalhamos bastante para criar exercícios de desenvolvimento de aplicativos em Android atuais. Você vai desenvolver aplicativos usando uma ampla variedade de tecnologias modernas. Todos os exercícios de pro-

gramação em Android exigem a implementação de aplicativos completos. Você vai ser convidado a melhorar os aplicativos existentes nos capítulos, a desenvolver aplicativos semelhantes, a usar sua criatividade para desenvolver seus próprios aplicativos utilizando as tecnologias abordadas no capítulo e a construir novos aplicativos com base nos aplicativos de código-fonte aberto disponíveis na Internet (e, novamente, **leia e obedeça os termos de licença de código-fonte aberto para cada aplicativo**). Os exercícios sobre Android incluem também perguntas a serem respondidas com uma resposta breve e do tipo verdadeiro/falso.

Nos exercícios sobre Java, será necessário lembrar termos e conceitos importantes; indicar o que os segmentos de código fazem; indicar o que está errado em uma parte do código; escrever instruções, métodos e classes Java; e escrever programas completos em Java.

Índice. Incluímos um índice abrangente para referência. O número de página da ocorrência de cada termo importante no livro está realçado no índice em **negrito**.

Software usado neste livro

Todo software que você vai precisar para este livro está disponível gratuitamente para download na Internet. Consulte a seção “Antes de começar” para ver os links de download.

Documentação. Toda documentação sobre Android e Java necessária para desenvolver aplicativos Android está disponível gratuitamente nos endereços <http://developer.android.com> e <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. A documentação do Eclipse está disponível no endereço www.eclipse.org/documentation. A documentação do Android Studio está disponível no endereço <http://developer.android.com/sdk/installing/studio.html>.

Recursos para professores

Os seguintes complementos (em inglês) estão disponíveis **somente para professores em** www.grupoa.com.br:

- **Slides para PowerPoint**[®] contendo todo o código e figuras do texto.
- **Arquivo de teste** com perguntas de resposta rápida.
- **Manual de soluções** com explicações para os **exercícios de resposta rápida** do final do capítulo, para conteúdo Java e Android. Para o conteúdo Java são fornecidas soluções para a *maioria* dos exercícios de programação.

Os exercícios de projeto de desenvolvimento de aplicativo Android sugeridos *não* são problemas do tipo “dever de casa” típicos. Eles tendem a ser projetos *de peso* – muitos dos quais podendo exigir semanas de esforço, possivelmente com os alunos trabalhando em equipes. **Soluções selecionadas** *só* são fornecidas para esses exercícios de projeto. Entre em contato conosco pelo endereço deitel@deitel.com, caso tenha dúvidas.

Para fazer download desse conteúdo, cadastre-se gratuitamente como professor, encontre e acesse a página do livro por meio do campo de busca e clique no link Material para o Professor. Não nos escreva solicitando acesso ao conteúdo para professores se você não for professor – ele é de uso restrito.

Antes de começar

Para obter informações sobre a configuração de seu computador para que você possa desenvolver aplicativos com Java e Android, consulte a seção “Antes de começar”, após este Prefácio.

Agradecimentos

Agradecemos a Barbara Deitel pelas longas horas dedicadas a este projeto – ela criou todos os nossos Java e Android Resource Centers e pesquisou centenas de detalhes técnicos pacientemente.

Este livro foi um trabalho de colaboração entre as divisões acadêmicas e profissionais da Pearson. Valorizamos a orientação, o bom senso e a energia de Tracy Johnson, editora executiva de Ciência da Computação. Tracy e sua equipe cuidam de todos os nossos livros acadêmicos. Carole Snyder recrutou os revisores acadêmicos do livro e gerenciou o processo de revisão. Bob Engelhardt gerenciou a publicação do livro. Escolhemos a arte da capa, e Marta Samsel a projetou.

Agradecemos também os esforços e aconselhamento de nosso amigo e colega de profissão Mark L. Taub, editor-chefe do Pearson Technology Group. Mark e sua equipe cuidaram de todos os nossos livros profissionais e produtos de vídeo LiveLessons. Kim Boedigheimer recrutou e gerenciou os revisores profissionais do conteúdo Android. John Fuller gerencia a produção de todos os nossos livros da Deitel Developer Series.

Gostaríamos de agradecer a Michael Morgano, antigo colega nosso na Deitel & Associates, Inc., agora desenvolvedor de Android na ImerjTM, que foi coautor das primeiras edições deste livro e de nosso livro *iPhone for Programmers: An App-Driven Approach*. Michael é um desenvolvedor de software extraordinariamente talentoso.

Revisores do conteúdo das edições recentes deste livro e do Android para Programadores: uma abordagem baseada em aplicativos

Agradecemos os esforços de nossos revisores da primeira e da segunda edição. Eles examinaram atentamente o texto e o código, dando muitas sugestões para melhorar a apresentação: Paul Beusterien (dirigente da Mobile Developer Solutions), Eric J. Bowden (diretor da Safe Driving Systems, LLC), Tony Cantrell (Georgia Northwestern Technical College), Ian G. Clifton (prestador de serviços e desenvolvedor de aplicativos Android), Daniel Galpin (defensor do Android e autor do livro *Intro to Android Application Development*), Jim Hathaway (desenvolvedor de aplicativos da Kellogg Company), Douglas Jones (engenheiro de software sênior da Fullpower Technologies), Charles Lasky (Nagautuck Community College), Enrique Lopez-Manas (principal arquiteto de Android da Sixt e professor de ciência da computação na Universidade de Alcalá em Madrid), Sebastian Nykopp (arquiteto-chefe da Reaktor), Michael Pardo (desenvolvedor de Android da Mobiata), Ronan “Zero” Schwarz (diretor de informação da OpenIntents), Arijit Sengupta (Wright State University), Donald Smith (Columbia College), Jesus Ubaldo Quevedo-Torrero (Universidade de Wisconsin, Parkside), Dawn Wick (Southwestern Community College) e Frank Xu (Gannon University).

Revisores do conteúdo das edições recentes do livro Java How to Program

Lance Andersen (Oracle), Soundararajan Angusamy (Sun Microsystems), Joseph Bowbeer (consultor), William E. Duncan (Louisiana State University), Diana Franklin (University of California, Santa Barbara), Edward F. Gehringer (North Carolina State University),

Huiwei Guan (Northshore Community College), Ric Heishman (George Mason University), Dr. Heinz Kabutz (JavaSpecialists.eu), Patty Kraft (San Diego State University), Lawrence Premkumar (Sun Microsystems), Tim Margush (University of Akron), Sue McFarland Metzger (Villanova University), Shyamal Mitra (The University of Texas at Austin), Peter Pilgrim (consultor), Manjeet Rege, Ph.D. (Rochester Institute of Technology), Manfred Riem (defensor do Java, consultor, Robert Half), Simon Ritter (Oracle), Susan Rodger (Duke University), Amr Sabry (Indiana University), José Antonio González Seco (Parlamento de Andalusia), Sang Shin (Sun Microsystems), S. Sivakumar (Astra Infotech Private Limited), Raghavan “Rags” Srinivas (Intuit), Monica Sweat (Georgia Tech), Vinod Varma (Astra Infotech Private Limited) e Alexander Zuev (Sun Microsystems).

Comentários, críticas e sugestões de melhoria do texto são bem-vindos. Envie correspondências diretamente para os autores para:

`deitel@deitel.com`

Ou para a editora da edição brasileira deste livro para:

`secretariaeditorial@grupoa.com.br`

Responderemos prontamente. Gostamos muito de escrever este livro – esperamos que você goste de lê-lo!

Paul Deitel

Harvey Deitel

Abbey Deitel

Os autores

Paul Deitel, diretor-executivo e diretor técnico-chefe da Deitel & Associates, Inc., é formado pelo MIT, onde estudou Tecnologia da Informação. Com a Deitel & Associates, Inc., ministrou centenas de cursos de programação para clientes de todo o mundo, incluindo Cisco, IBM, Siemens, Sun Microsystems, Dell, Fidelity, NASA (no Centro Espacial Kennedy), National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, SunGard Higher Education, Nortel Networks, Puma, iRobot, Invensys e muitas mais. Ele e seu coautor, Dr. Harvey M. Deitel, são os autores de livros profissionais e acadêmicos e vídeos sobre linguagens de programação mais vendidos no mundo.

Dr. Harvey Deitel, presidente do conselho administrativo e estrategista-chefe da Deitel & Associates, Inc., tem mais de 50 anos de experiência na área da computação. É bacharel e mestre em engenharia elétrica pelo MIT e doutor em matemática pela Universidade de Boston. Tem ampla experiência em ensino superior, incluindo um período como professor e chefe do Departamento de Ciência da Computação do Boston College. As publicações da Deitel têm recebido reconhecimento internacional, com traduções publicadas em chinês, coreano, japonês, alemão, russo, espanhol, francês, polonês, italiano, português, grego, urdu e turco.

Abbey Deitel, presidente da Deitel & Associates, Inc., é formada pela Tepper School of Management da Carnegie Mellon University, onde fez bacharelado em Gerenciamento Industrial. Abbey gerencia as operações comerciais da Deitel & Associates, Inc. há 16 anos. Colaborou em inúmeras publicações da Deitel & Associates e, com Paul e Harvey, é coautora dos livros *Android para programadores: Uma abordagem baseada em aplicativos*, 2/e, *iPhone for Programmers: An App-Driven Approach*, *Internet & World Wide Web How to Program*, 5/e, *Visual Basic 2012 How to Program*, 6/e e *Simply Visual Basic 2010*, 5/e.

Treinamento corporativo da Deitel® Dive-Into® Series

A Deitel & Associates é uma empresa que atua no mercado editorial, de treinamento corporativo e de desenvolvimento de software, especializada em linguagens de programação, tecnologia de objetos, tecnologias de Internet e software web, além de desenvolvimento de aplicativos para iOS e Android. Seus clientes incluem as maiores empresas do mundo, órgãos governamentais, setores das forças armadas e instituições acadêmicas. Oferece cursos de treinamento sobre as principais linguagens e plataformas de programação – como desenvolvimento de aplicativos Android, desenvolvimento de aplicativos Objective-C e iOS, Java™, C++, Visual C++®, C, Visual C#®, Visual Basic®, XML®, Python®, tecnologia de objetos, programação para Internet e web e uma crescente lista de cursos de programação e desenvolvimento de software adicionais.

Há 37 anos, a Deitel & Associates, Inc. publica livros profissionais sobre programação, livros-texto acadêmicos e cursos em vídeo LiveLessons. Para saber mais sobre o currículo de treinamento corporativo Dive-Into® Series da Deitel, visite (em inglês) ou escreva para:

www.deitel.com/training

deitel@deitel.com

Esta página foi deixada em branco intencionalmente.



Antes de começar

Nesta seção, você vai configurar seu computador para usá-lo com este livro. As ferramentas de desenvolvimento para Android são atualizadas frequentemente. Antes de ler esta seção, consulte o site do livro (em inglês)

<http://www.deitel.com/books/AndroidHTP2/>

para ver se postamos uma versão atualizada.

Convenções de fonte e atribuição de nomes

Utilizamos fontes para diferenciar entre componentes de tela (como nomes de menu e itens de menu) e código ou comandos Java. Nossa convenção é mostrar componentes de tela com a fonte **Helvetica** sem serifa e em negrito (por exemplo, menu **Project**), e nomes de arquivo, código e comandos Java na fonte **Lucida** sem serifa (por exemplo, a palavra-chave `public` ou a classe `Activity`). Ao especificarmos comandos para selecionar em menus, usamos a notação **>** para indicar um item de menu a ser selecionado. Por exemplo, **Window > Preferences** indica que você deve selecionar o item de menu **Preferences** no menu **Window**.

Requisitos de sistema para software e hardware

Para desenvolver aplicativos Android, você precisa de um sistema Windows®, Linux ou Mac OS X. Para ver os requisitos mais recentes do sistema operacional, visite

<http://developer.android.com/sdk/index.html>

e procure o cabeçalho **SYSTEM REQUIREMENTS**. Desenvolvemos os aplicativos deste livro usando os seguintes programas:

- Java SE 7 Software Development Kit
- Android SDK/ADT Bundle baseado no IDE Eclipse
- SDK do Android versões 4.3 e 4.4

Você vai ver como obter cada um deles nas próximas seções.

Instalando o JDK (Java Development Kit)

O Android exige o *JDK (Java Development Kit)* versão 7 (JDK 7) ou 6 (JDK 6). Usamos o *JDK 7*. Para baixar o JDK para Windows, OS X ou Linux, acesse

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Você só precisa do JDK. Escolha a versão de 32 ou de 64 bits, de acordo com o hardware e o sistema operacional de seu computador. Os computadores mais recentes têm hardware de 64 bits – verifique as especificações de seu sistema. Se você tem um sistema operacional de 32 bits, deve usar o JDK de 32 bits. Siga as instruções de instalação em

<http://docs.oracle.com/javase/7/docs/webnotes/install/index.html>

Opções de IDE (Integrated Development Environment) para Android

Agora o Google fornece duas opções de IDE para Android:

- Android SDK/ADT bundle – uma versão do *IDE Eclipse* que vem previamente configurada com o SDK (Software Development Kit) do Android e o plug-in ADT (Android Development Tools) mais recentes. Quando este livro estava sendo produzido, eles eram o SDK do Android versão 4.4 e ADT versão 22.3.
- Android Studio – o novo IDE do Android do Google, baseado no IntelliJ® IDEA e seu IDE futuro preferido.

O Android SDK/ADT bundle tem sido amplamente usado em desenvolvimento de aplicativos Android há vários anos. O Android Studio, apresentado em maio de 2013, é uma *versão prévia de teste* e está evoluindo rapidamente. Por isso, no livro, ficaremos com o amplamente usado Android SDK/ADT bundle e, como complementos online (em inglês), em

<http://www.deitel.com/books/AndroidHTP2>

forneceremos versões para Android Studio da seção “Teste do Capítulo 1” (Chapter 1 Test-Drive) e da seção “Construção da interface gráfica de usuário” (Building the GUI) de cada aplicativo, conforme for apropriado.

Instalando o Android SDK/ADT bundle

O IDE SDK/ADT bundle, que inclui o Eclipse e inúmeras outras ferramentas de desenvolvimento, não está mais disponível para download no site developer.android.com. Agora, o Android Studio é o IDE preferido pelo Google para o desenvolvimento de aplicativos Android. Este livro está todo baseado em Eclipse. Muitos leitores relataram conseguir adaptar facilmente as instruções do Eclipse e realizar todos os exercícios deste livro no Android Studio. Mas, se você quiser reproduzir exatamente as mesmas experiências apresentadas neste livro, preparamos um documento que explica como baixar e instalar o Eclipse e as ferramentas Android separadamente e então integrá-las. Acesse

www.grupoa.com.br

Cadastre-se gratuitamente, encontre e acesse a página do livro por meio do campo de busca e clique no link Conteúdo Online para fazer download do documento.

Instalando o Android Studio

As instruções do IDE no livro impresso usam o Android SDK/ADT bundle. Opcionalmente, você também pode instalar e usar o Android Studio. Para baixar o Android Studio, acesse

<http://developer.android.com/sdk/installing/studio.html>

e clique no botão **Download Android Studio**. Quando o download terminar, execute o instalador e siga as instruções na tela para concluir a instalação. [*Obs.:* Para desenvolvimento com Android 4.4 no Android Studio, agora o Android suporta recursos da linguagem Java SE 7, incluindo o operador diamante, captura múltipla (multi-catch), componentes String na instrução switch e try-with-resources.]

Configure o nível de compatibilidade com o compilador Java e exiba números de linha

O Android não suporta Java SE 7 completamente. Para garantir que os exemplos do livro sejam compilados corretamente, configure o Eclipse para produzir arquivos compatíveis com Java SE 6, executando os passos a seguir:

1. Abra o Eclipse (🔗 ou 🌐), que está localizado na subpasta eclipse da pasta de instalação do Android SDK/ADT bundle.
2. Quando a janela **Workspace Launcher** aparecer, clique em **OK**.
3. Selecione **Window > Preferences** para exibir a janela **Preferences**. No Mac OS X, selecione **ADT > Preferences....**
4. Expanda o nó **Java** e selecione o nó **Compiler**. Sob **JDK Compliance**, configure **Compiler compliance level** como 1.6 (para indicar que o Eclipse deve produzir código compilado compatível com o Java SE 6).
5. Expanda o nó **General > Editors** e selecione **TextEditors**; em seguida, certifique-se de que **Show line numbers** esteja selecionado e clique em **OK**.
6. Feche o Eclipse.

SDK do Android 4.3

Os exemplos deste livro foram escritos com os SDKs do Android 4.3 e 4.4. Quando esta obra estava sendo produzida, 4.4 era a versão incluída com o Android SDK/ADT bundle e com o Android Studio. Você também deve instalar o Android 4.3 (e outras versões para as quais queira dar suporte em seus aplicativos). Para instalar outras versões da plataforma Android, execute os passos a seguir (pulando os passos 1 e 2 se o Eclipse já estiver aberto):

1. Abra o Eclipse. Dependendo de sua plataforma, o ícone aparecerá como 🔗 ou 🌐.
2. Quando a janela **Workspace Launcher** aparecer, clique em **OK**.
3. No Mac OS X, se vir uma janela indicando “Could not find SDK folder ‘/Users/SuaConta/android-sdk-macosx/’”, clique em **Open Preferences**, depois em **Browse...** e selecione a pasta sdk localizada no lugar onde você extraiu o Android SDK/ADT bundle.
4. Selecione **Window > Android SDK Manager** para exibir o **Android SDK Manager** (Fig. 1).

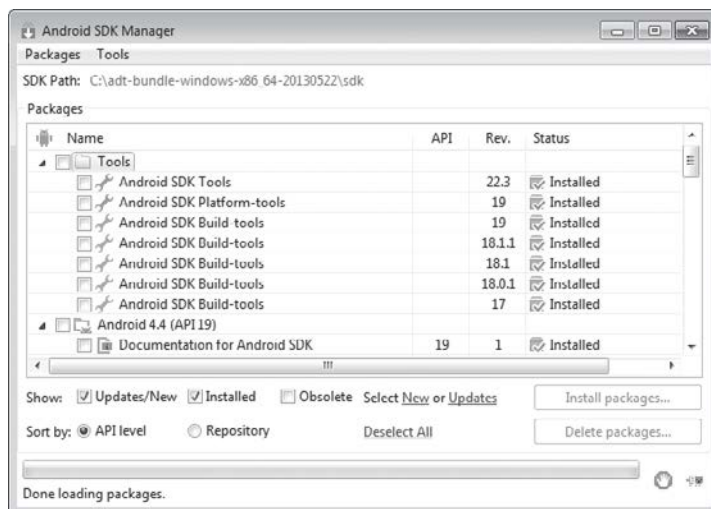


Figura 1 Janela Android SDK Manager.

5. A coluna **Name** do **Android SDK Manager** mostra todas as ferramentas, versões de plataforma e extras (por exemplo, as APIs para interagir com serviços do Google, como o Maps) que podem ser instaladas. Desmarque a caixa de seleção **Installed**. Em seguida, se **Tools**, **Android 4.4 (API19)**, **Android 4.3 (API18)** e **Extras** aparecerem na lista **Packages**, certifique-se de que estejam marcados e clique em **Install # packages...** (# é o número de itens a serem instalados) para exibir a janela **Choose Packages to Install**. A maioria dos itens no nó **Extras** é opcional. Para este livro, você precisará de **Android Support Library** e de **Google Play services**. O item **Google USB Driver** é necessário para usuários de Windows que queiram testar aplicativos em dispositivos Android.
6. Na janela **Choose Packages to Install**, leia os acordos de licenciamento de cada item. Quando tiver terminado, clique no botão de opção **Accept License e**, em seguida, clique no botão **Install**. O status do processo de instalação aparecerá na janela **Android SDK Manager**.

Criando AVDs (Android Virtual Devices)

O **emulador do Android**, incluído no SDK do Android, permite testar aplicativos em seu computador em vez de no dispositivo Android real. Isso é útil se você está aprendendo Android e não tem acesso a dispositivos Android, mas pode ser *muito* lento, de modo que um dispositivo real é preferível, caso disponha de um. Existem alguns recursos de aceleração de hardware que podem melhorar o desempenho do emulador (developer.android.com/tools/devices/emulator.html#acceleration). Antes de executar um aplicativo no emulador, você precisa criar um **AVD (Android Virtual Device ou Dispositivo Android Virtual)**, o qual define as características do dispositivo no qual se deseja fazer o teste, incluindo o tamanho da tela em pixels, a densidade de pixels, o tamanho físico da tela, o tamanho do cartão SD para armazenamento de dados e muito mais. A fim de testar seus aplicativos para vários dispositivos Android, você pode criar AVDs que emulem exclusivamente cada dispositivo. Para este livro, usamos AVDs para dispositivos Android de referência do Google – o telefone Nexus 4, o tablet Nexus 7 pequeno e o tablet Nexus 10 grande –, os quais executam em versões sem modificação do Android. Para isso, execute os passos a seguir:

1. Abra o Eclipse.
2. Selecione **Window > Android Virtual Device Manager** para exibir a janela **Android Virtual Device Manager** e, então, selecione a guia **Device Definitions** (Fig. 2).

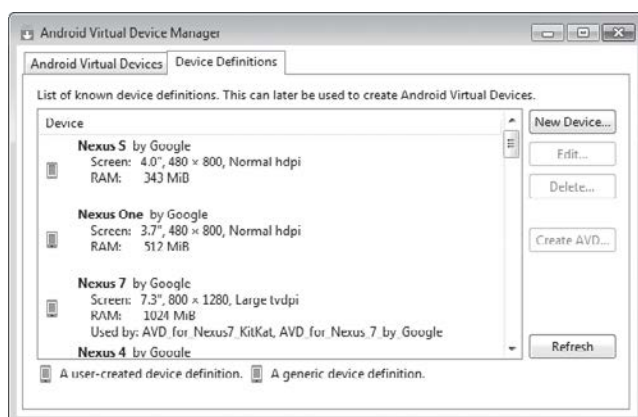


Figura 2 Janela **Android Virtual Device Manager**.

3. O Google fornece dispositivos previamente configurados que podem ser usados para criar AVDs. Selecione **Nexus 4 by Google** e clique em **Create AVD...** para exibir a janela **Create new Android Virtual Device (AVD)** (Fig. 3). Em seguida, configure as opções como mostrado e clique em **OK** para criar o AVD. Se você marcar **Hardware keyboard present**, poderá usar o teclado de seu computador para digitar dados em aplicativos que estiverem executando no AVD, mas isso pode impedir que o teclado virtual apareça na tela. Se seu computador não tem câmera, você pode selecionar **Emulated** para as opções **Front Camera** e **Back Camera**. Cada AVD criado tem muitas outras opções especificadas em seu arquivo `config.ini`. Esse arquivo pode ser modificado conforme descrito em

<http://developer.android.com/tools/devices/managing-avds.html>

para corresponder mais precisamente à configuração de hardware de seu dispositivo.

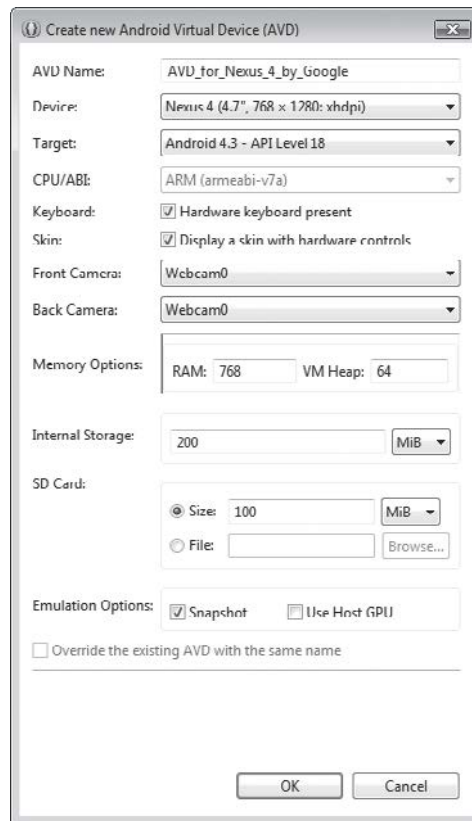


Figura 3 Configurando um AVD do smartphone Nexus 4 para Android 4.3.

4. Também configuramos AVDs para Android 4.3 que representam o Nexus 7 e o Nexus 10 do Google, para testar nossos aplicativos de tablet. Suas configurações aparecem na Fig. 4. Além disso, configuramos AVDs do Android 4.4 para Nexus 4, Nexus 7 e Nexus 10 com os nomes: `AVD_for_Nexus_4_KitKat`, `AVD_for_Nexus_7_KitKat` e `AVD_for_Nexus_10_KitKat`.

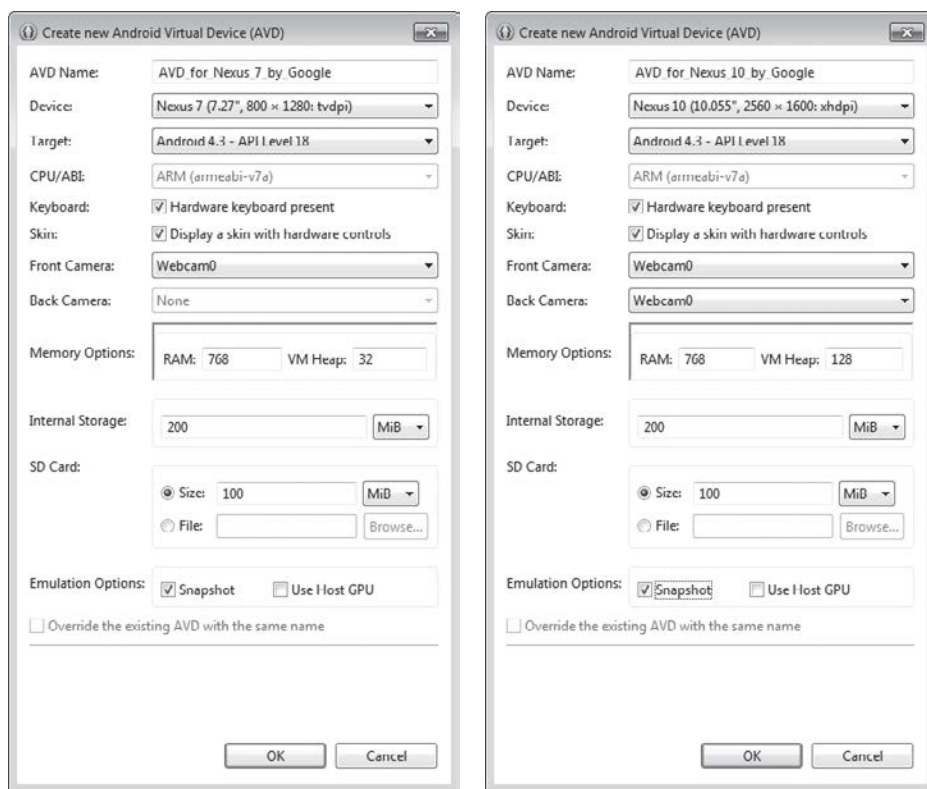


Figura 4 Configurando AVDs para tablets Nexus 7 e Nexus 10.

(Opcional) Configurando um dispositivo Android para desenvolvimento

Conforme mencionamos, testar aplicativos em AVDs pode ser lento, devido ao desempenho do AVD. Caso você disponha de um dispositivo Android, deve testar os aplicativos nesse dispositivo. Além disso, existem alguns recursos que só podem ser testados em dispositivos reais. Para executar seus aplicativos em dispositivos Android, siga as instruções que se encontram em

<http://developer.android.com/tools/device.html>

Caso esteja desenvolvendo no Microsoft Windows, você também precisará do driver Windows USB para dispositivos Android. Em alguns casos, no Windows, talvez também seja necessário drivers USB específicos do dispositivo. Para ver uma lista de sites de driver USB para várias marcas de dispositivo, visite:

<http://developer.android.com/tools/extras/oem-usb.html>

Obtendo os exemplos de código do livro

Os exemplos do livro estão disponíveis para download em

www.grupoa.com.br

Caso ainda não tenha registro em nosso site, acesse www.deitel.com e clique no link **Register**. Preencha suas informações. O registro é gratuito e não compartilhamos suas informações. Verifique se o endereço de e-mail de seu registro foi digitado corretamente – você receberá um e-mail de confirmação com seu código de verificação. *Você deve clicar no link de verificação no e-mail antes de poder se inscrever em www.deitel.com pela primeira vez.* Configure seu cliente de e-mail para permitir e-mails de [deitel.com](http://www.deitel.com) a fim de garantir que o e-mail de verificação não seja filtrado como correspondência não desejada. Enviamos apenas e-mails de gerenciamento de conta ocasionais, a não ser que você se registre separadamente em nossa newsletter *Deitel® Buzz Online* por e-mail em

<http://www.deitel.com/newsletter/subscribe.html>

Em seguida, visite www.deitel.com e inscreva-se usando o link **Login** abaixo de nosso logotipo, no canto superior esquerdo da página. Acesse <http://www.deitel.com/books/AndroidHTP2/>. Clique no link **Examples** para baixar em seu computador um arquivo ZIP compactado, contendo os exemplos. Clique duas vezes no arquivo ZIP para descompactá-lo e tome nota de onde você extraiu o conteúdo do arquivo em seu sistema.

Um lembrete sobre as ferramentas de desenvolvimento para Android

O Google atualiza *frequentemente* as ferramentas de desenvolvimento para Android. Muitas vezes, isso causa problemas na compilação de nossos aplicativos, quando, na verdade, os aplicativos não contêm erros. Se você importar um de nossos aplicativos para o Eclipse ou para o Android Studio e ele não compilar, provavelmente há um problema de configuração secundário. Entre em contato conosco por e-mail em deitel@deitel.com ou postando uma pergunta no:

- Facebook® – [facebook.com/DeitelFan](https://www.facebook.com/DeitelFan)
- Google+™ – [google.com/+DeitelFan](https://plus.google.com/+DeitelFan)

e o ajudaremos a resolver o problema.

Agora você já instalou todo o software e baixou os exemplos de código que precisará para estudar o desenvolvimento de aplicativos Android com este livro e para começar a desenvolver seus próprios aplicativos. Aproveite!

Esta página foi deixada em branco intencionalmente.



Sumário

I	Introdução ao Android	I
1.1	Introdução	2
1.2	Android – o sistema operacional móvel líder mundial	3
1.3	Recursos do Android	3
1.4	Sistema operacional Android	7
1.4.1	Android 2.2 (Froyo)	7
1.4.2	Android 2.3 (Gingerbread)	8
1.4.3	Android 3.0 a 3.2 (Honeycomb)	8
1.4.4	Android 4.0 a 4.0.4 (Ice Cream Sandwich)	9
1.4.5	Android 4.1 a 4.3 (Jelly Bean)	10
1.4.6	Android 4.4 (KitKat)	10
1.5	Baixando aplicativos do Google Play	11
1.6	Pacotes	12
1.7	O SDK do Android	13
1.8	Programação orientada a objetos: uma breve recapitulação	16
1.8.1	O automóvel como um objeto	16
1.8.2	Métodos e classes	17
1.8.3	Instanciação	17
1.8.4	Reutilização	17
1.8.5	Mensagens e chamadas de método	17
1.8.6	Atributos e variáveis de instância	18
1.8.7	Encapsulamento	18
1.8.8	Herança	18
1.8.9	Análise e projeto orientados a objetos	18
1.9	Teste do aplicativo Doodlz em um AVD	19
1.9.1	Executando o aplicativo Doodlz no AVD do smartphone Nexus 4	19
1.9.2	Executando o aplicativo Doodlz no AVD de um tablet	28
1.9.3	Executando o aplicativo Doodlz em um aparelho Android	29
1.10	Construção de excelentes aplicativos Android	30
1.11	Recursos para desenvolvimento com Android	32
1.12	Para finalizar	34
2	Aplicativo Welcome	37
2.1	Introdução	38
2.2	Visão geral das tecnologias	39
2.2.1	IDE Android Developer Tools	39
2.2.2	Componentes <code>TextView</code> e <code>ImageView</code>	39
2.2.3	Recursos do aplicativo	39
2.2.4	Acessibilidade	39
2.2.5	Internacionalização	40

2.3	Criação de um aplicativo	40
2.3.1	Ativação do IDE Android Developer Tools	40
2.3.2	Criação de um novo projeto	40
2.3.3	Caixa de diálogo New Android Application	40
2.3.4	Passo Configure Project	42
2.3.5	Passo Configure Launcher Icon	43
2.3.6	Passo Create Activity	44
2.3.7	Passo Blank Activity	45
2.4	Janela Android Developer Tools	45
2.4.1	Janela Package Explorer	46
2.4.2	Janelas do editor	47
2.4.3	Janela Outline	47
2.4.4	Arquivos de recurso do aplicativo	47
2.4.5	Editor Graphical Layout	48
2.4.6	A interface gráfica de usuário padrão	48
2.5	Construção da interface gráfica de usuário do aplicativo com o editor Graphical Layout	49
2.5.1	Adição de imagens ao projeto	50
2.5.2	Alteração da propriedade Id dos componentes RelativeLayout e TextView	51
2.5.3	Adição e configuração do componente TextView	52
2.5.4	Adição de componentes ImageView para exibir as imagens	56
2.6	Execução do aplicativo Welcome	58
2.7	Torne seu aplicativo acessível	59
2.8	Internacionalização de seu aplicativo	61
2.9	Para finalizar	65

3 Aplicativo Tip Calculator **68**

3.1	Introdução	69
3.2	Teste do aplicativo Tip Calculator	70
3.3	Visão geral das tecnologias	71
3.3.1	Classe Activity	71
3.3.2	Métodos de ciclo de vida de Activity	71
3.3.3	Organização de componentes de visualização com LinearLayout e GridLayout	72
3.3.4	Criação e personalização da interface gráfica do usuário com o editor Graphical Layout e com as janelas Outline e Properties	73
3.3.5	Formatação de números como moeda corrente específica da localidade e strings de porcentagem	73
3.3.6	Implementação da interface TextWatcher para lidar com alterações de texto em componente EditText	74
3.3.7	Implementação da interface OnSeekBarChangeListener para lidar com alterações na posição do cursor no componente SeekBar	74
3.3.8	AndroidManifest.xml	74
3.4	Construção da interface gráfica do usuário do aplicativo	74
3.4.1	Introdução ao componente GridLayout	74
3.4.2	Criação do projeto TipCalculator	76
3.4.3	Alteração para um componente GridLayout	77

3.4.4	Adição dos componentes TextView, EditText, SeekBar e LinearLayout	77
3.4.5	Personalização das visualizações para concluir o projeto	80
3.5	Adição de funcionalidade ao aplicativo	84
3.6	AndroidManifest.xml	91
3.7	Para finalizar	93

4 Aplicativo Twitter® Searches 98

4.1	Introdução	99
4.2	Teste do aplicativo	100
4.2.1	Importação e execução do aplicativo	100
4.2.2	Adição de uma busca favorita	101
4.2.3	Visualização dos resultados de uma busca no Twitter	102
4.2.4	Edição de uma pesquisa	102
4.2.5	Compartilhamento de uma pesquisa	104
4.2.6	Exclusão de uma pesquisa	105
4.2.7	Rolagem por pesquisas salvas	105
4.3	Visão geral das tecnologias	106
4.3.1	ListView	106
4.3.2	ListActivity	106
4.3.3	Personalização do layout de um componente ListActivity	107
4.3.4	ImageButton	107
4.3.5	SharedPreferences	107
4.3.6	Objetos Intent para ativar outras atividades	108
4.3.7	AlertDialog	108
4.3.8	AndroidManifest.xml	109
4.4	Construção da interface gráfica do usuário do aplicativo	109
4.4.1	Criação do projeto	109
4.4.2	Visão geral de activity_main.xml	110
4.4.3	Adição de GridLayout e componentes	111
4.4.4	Barra de ferramentas do editor Graphical Layout	116
4.4.5	Layout do item ListView: list_item.xml	117
4.5	Construção da classe MainActivity	118
4.5.1	As instruções package e import	118
4.5.2	Extensão de ListActivity	120
4.5.3	Campos da classe MainActivity	120
4.5.4	Sobrescrevendo o método onCreate de Activity	121
4.5.5	Classe interna anônima que implementa a interface OnClickListener de saveButton para salvar uma pesquisa nova ou atualizada	123
4.5.6	Método addTaggedSearch	125
4.5.7	Classe interna anônima que implementa a interface onItemClick de ListView para exibir resultados de pesquisa	126
4.5.8	Classe interna anônima que implementa a interface onItemLongClickListener de ListView para compartilhar, editar ou excluir uma pesquisa	128
4.5.9	Método shareSearch	130
4.5.10	Método deleteSearch	131
4.6	AndroidManifest.xml	133
4.7	Para finalizar	133

5	Aplicativo Flag Quiz	137
5.1	Introdução	138
5.2	Teste do aplicativo Flag Quiz	140
5.2.1	Importação e execução do aplicativo	140
5.2.2	Configuração do teste	140
5.2.3	O teste	142
5.3	Visão geral das tecnologias	143
5.3.1	Menus	143
5.3.2	Fragmentos	144
5.3.3	Métodos do ciclo de vida de um fragmento	144
5.3.4	Gerenciamento de fragmentos	145
5.3.5	Preferências	145
5.3.6	Pasta assets	145
5.3.7	Pastas de recurso	146
5.3.8	Suporte para diferentes tamanhos e resoluções de tela	146
5.3.9	Determinação do tamanho da tela	147
5.3.10	Componentes Toast para exibir mensagens	147
5.3.11	Uso de um objeto Handler para executar um objeto Runnable no futuro	147
5.3.12	Aplicação de uma animação a um objeto View	147
5.3.13	Registro de mensagens de exceção	148
5.3.14	Uso de um objeto Intent explícito para ativar outra atividade no mesmo aplicativo	148
5.3.15	Estruturas de dados em Java	148
5.4	Construção da interface gráfica do usuário e do arquivo de recursos	148
5.4.1	Criação do projeto	148
5.4.2	strings.xml e recursos de String formatados	149
5.4.3	arrays.xml	150
5.4.4	colors.xml	151
5.4.5	dimens.xml	151
5.4.6	Layout de activity_settings.xml	152
5.4.7	Layout de activity_main.xml para orientação retrato para telefones e tablets	152
5.4.8	Layout de fragment_quiz.xml	153
5.4.9	Layout de activity_main.xml para orientação paisagem para tablets	155
5.4.10	Arquivo preferences.xml para especificar as configurações do aplicativo	156
5.4.11	Criação da animação da bandeira	157
5.5	Classe MainActivity	159
5.5.1	Instrução package, instruções import e campos	159
5.5.2	Método sobrescrito onCreate de Activity	160
5.5.3	Método sobrescrito onStart de Activity	161
5.5.4	Método sobrescrito onCreateOptionsMenu de Activity	162
5.5.5	Método sobrescrito onOptionsItemSelected de Activity	163
5.5.6	Classe interna anônima que implementa OnSharedPreferenceChangeListener	163

5.6	Classe QuizFragment	165
5.6.1	A instrução package e as instruções import	165
5.6.2	Campos	166
5.6.3	Método sobrescrito onCreateView de Fragment	167
5.6.4	Método updateGuessRows	168
5.6.5	Método updateRegions	169
5.6.6	Método resetQuiz	169
5.6.7	Método loadNextFlag	171
5.6.8	Método getCountryName	173
5.6.9	Classe interna anônima que implementa OnClickListener	173
5.6.10	Método disableButtons	175
5.7	Classe SettingsFragment	176
5.8	Classe SettingsActivity	176
5.9	AndroidManifest.xml	177
5.10	Para finalizar	177

6 **Aplicativo Cannon Game** **182**

6.1	Introdução	183
6.2	Teste do aplicativo Cannon Game	185
6.3	Visão geral das tecnologias	185
6.3.1	Anexação de um componente View personalizado a um layout	185
6.3.2	Uso da pasta de recurso raw	185
6.3.3	Métodos de ciclo de vida de Activity e Fragment	185
6.3.4	Sobrescrevendo o método onTouchEvent de View	186
6.3.5	Adição de som com SoundPool e AudioManager	186
6.3.6	Animação quadro a quadro com Threads, SurfaceView e SurfaceHolder	186
6.3.7	Deteção de colisão simples	187
6.3.8	Desenho de elementos gráficos com Paint e Canvas	187
6.4	Construção da interface gráfica do usuário e arquivos de recurso do aplicativo	187
6.4.1	Criação do projeto	187
6.4.2	strings.xml	188
6.4.3	fragment_game.xml	188
6.4.4	activity_main.xml	189
6.4.5	Adição dos sons ao aplicativo	189
6.5	A classe Line mantém os extremos de uma linha	189
6.6	Subclasse MainActivity de Activity	190
6.7	Subclasse CannonGameFragment de Fragment	191
6.8	Subclasse CannonView de View	192
6.8.1	As instruções package e import	192
6.8.2	Variáveis de instância e constantes	193
6.8.3	Construtor	194
6.8.4	Sobrescrevendo o método onSizeChanged de View	196
6.8.5	Método newGame	197
6.8.6	Método updatePositions	198
6.8.7	Método fireCannonball	201
6.8.8	Método alignCannon	202

6.8.9 Método drawGameElements	202
6.8.10 Método showGameOverDialog	204
6.8.11 Métodos stopGame e releaseResources	205
6.8.12 Implementando os métodos de SurfaceHolder.Callback	206
6.8.13 Sobrescrevendo o método onTouchEvent de View	207
6.8.14 CannonThread: usando uma thread para criar um loop de jogo	208
6.9 Para finalizar	209

7 Aplicativo Doodlz 215

7.1 Introdução	216
7.2 Visão geral das tecnologias	218
7.2.1 Uso de SensorManager para detectar eventos de acelerômetro	218
7.2.2 Componentes DialogFragment personalizados	218
7.2.3 Desenho com Canvas e Bitmap	219
7.2.4 Processamento de múltiplos eventos de toque e armazenamento de linhas em objetos Path	219
7.2.5 Modo imersivo do Android 4.4	219
7.2.6 GestureDetector e SimpleOnGestureListener	219
7.2.7 Salvando o desenho na galeria do dispositivo	219
7.2.8 Impressão no Android 4.4 e a classe PrintHelper da Android Support Library	220
7.3 Construção da interface gráfica do usuário e arquivos de recurso do aplicativo	220
7.3.1 Criação do projeto	220
7.3.2 strings.xml	221
7.3.3 dimens.xml	221
7.3.4 Menu do componente DoodleFragment	222
7.3.5 Layout de activity_main.xml para MainActivity	223
7.3.6 Layout de fragment_doodle.xml para DoodleFragment	223
7.3.7 Layout de fragment_color.xml para ColorDialogFragment	224
7.3.8 Layout de fragment_line_width.xml para LineWidthDialogFragment	226
7.3.9 Adição da classe EraseImageDialogFragment	227
7.4 Classe MainActivity	228
7.5 Classe DoodleFragment	229
7.6 Classe DoodleView	235
7.7 Classe ColorDialogFragment	247
7.8 Classe LineWidthDialogFragment	250
7.9 Classe EraseImageDialogFragment	253
7.10 Para finalizar	255

8 Aplicativo Address Book 259

8.1 Introdução	260
8.2 Teste do aplicativo Address Book	263
8.3 Visão geral das tecnologias	264
8.3.1 Exibição de fragmentos com componentes FragmentTransaction	264
8.3.2 Comunicação de dados entre um fragmento e uma atividade hospedeira	264
8.3.3 Método onSaveInstanceState	264

8.3.4	Definindo estilos e aplicando-os nos componentes da interface gráfica do usuário	265
8.3.5	Especificação de um fundo para um componente <code>TextView</code>	265
8.3.6	Extensão da classe <code>ListFragment</code> para criar um fragmento contendo um componente <code>ListView</code>	265
8.3.7	Manipulação de um banco de dados <code>SQLite</code>	265
8.3.8	Execução de operações de banco de dados fora da thread da interface gráfica do usuário com elementos <code>AsyncTask</code>	265
8.4	Construção da interface gráfica do usuário e do arquivo de recursos	266
8.4.1	Criação do projeto	266
8.4.2	Criação das classes do aplicativo	266
8.4.3	<code>strings.xml</code>	267
8.4.4	<code>styles.xml</code>	267
8.4.5	<code>textview_border.xml</code>	268
8.4.6	Layout de <code>MainActivity</code> : <code>activity_main.xml</code>	269
8.4.7	Layout de <code>DetailsFragment</code> : <code>fragment_details.xml</code>	270
8.4.8	Layout de <code>AddEditFragment</code> : <code>fragment_add_edit.xml</code>	271
8.4.9	Definição dos menus dos fragmentos	272
8.5	Classe <code>MainActivity</code>	273
8.6	Classe <code>ContactListFragment</code>	279
8.7	Classe <code>AddEditFragment</code>	286
8.8	Classe <code>DetailsFragment</code>	291
8.9	Classe utilitária <code>DatabaseConnector</code>	298
8.10	Para finalizar	304

9 Google Play e questões de comercialização de aplicativos **308**

9.1	Introdução	309
9.2	Preparação dos aplicativos para publicação	309
9.2.1	Teste do aplicativo	310
9.2.2	Acordo de Licença de Usuário Final	310
9.2.3	Ícones e rótulos	310
9.2.4	Controlando a versão de seu aplicativo	311
9.2.5	Licenciamento para controle de acesso a aplicativos pagos	311
9.2.6	Ofuscando seu código	311
9.2.7	Obtenção de uma chave privada para assinar digitalmente seu aplicativo	312
9.2.8	Capturas de tela	312
9.2.9	Vídeo promocional do aplicativo	314
9.3	Precificação de seu aplicativo: gratuito ou pago	314
9.3.1	Aplicativos pagos	315
9.3.2	Aplicativos gratuitos	315
9.4	Monetização de aplicativos com anúncio incorporado	316
9.5	Monetização de aplicativos: utilização de cobrança incorporada para vender bens virtuais	317
9.6	Registro no Google Play	318
9.7	Abertura de uma conta no Google Wallet	319

9.8	Carregamento de seus aplicativos no Google Play	320
9.9	Ativação do Play Store dentro de seu aplicativo	321
9.10	Gerenciamento de seus aplicativos no Google Play	322
9.11	Outras lojas de aplicativos Android	322
9.12	Outras plataformas populares de aplicativos móveis	323
9.13	Comercialização de aplicativos	323
9.14	Para finalizar	327

A Introdução aos aplicativos Java **331**

A.1	Introdução	332
A.2	Seu primeiro programa em Java: impressão de uma linha de texto	332
A.3	Modificação de seu primeiro programa em Java	336
A.4	Exibição de texto com <code>printf</code>	338
A.5	Outro aplicativo: soma de valores inteiros	338
A.6	Conceitos sobre memória	342
A.7	Aritmética	343
A.8	Tomada de decisão: operadores de igualdade e relacionais	346
A.9	Para finalizar	350

B Introdução a classes, objetos, métodos e strings **355**

B.1	Introdução	356
B.2	Declaração de uma classe com um método e instanciação de um objeto de uma classe	356
B.3	Declaração de um método com um parâmetro	359
B.4	Variáveis de instância, métodos <i>set</i> e métodos <i>get</i>	362
B.5	Tipos primitivos <i>versus</i> tipos de referência	366
B.6	Inicialização de objetos com construtores	367
B.7	Números de ponto flutuante e o tipo <code>double</code>	369
B.8	Para finalizar	373

C Instruções de controle **377**

C.1	Introdução	378
C.2	Algoritmos	378
C.3	Pseudocódigo	379
C.4	Estruturas de controle	379
C.5	Instrução de seleção simples <code>if</code>	380
C.6	Instrução de seleção dupla <code>if...else</code>	380
C.7	Instrução de repetição <code>while</code>	383
C.8	Estudo de caso: repetição controlada por contador	383
C.9	Estudo de caso: repetição controlada por sentinela	387
C.10	Estudo de caso: instruções de controle aninhadas	392
C.11	Operadores de atribuição compostos	395
C.12	Operadores de incremento e decremento	395
C.13	Tipos primitivos	397
C.14	Fundamentos da repetição controlada por contador	398
C.15	Instrução de repetição <code>for</code>	398

C.16	Exemplos de uso da instrução <code>for</code>	400
C.17	Instrução de repetição <code>do...while</code>	403
C.18	Instrução de seleção múltipla <code>switch</code>	404
C.19	Instruções <code>break</code> e <code>continue</code>	410
C.20	Operadores lógicos	410
C.21	Para finalizar	413

D Métodos: uma investigação mais aprofundada 421

D.1	Introdução	422
D.2	Módulos de programa em Java	422
D.3	Métodos estáticos, campos estáticos e a classe <code>Math</code>	423
D.4	Declaração de métodos com vários parâmetros	425
D.5	Observações sobre declaração e uso de métodos	428
D.6	Pilha de chamada de métodos e registros de ativação	429
D.7	Promoção e conversão de argumentos	429
D.8	Pacotes da API Java	431
D.9	Introdução à geração de números aleatórios	432
D.9.1	Escala e deslocamento de números aleatórios	433
D.9.2	Repetitividade de números aleatórios para teste e depuração	434
D.10	Estudo de caso: um jogo de azar – introdução a enumerações	434
D.11	Escopo das declarações	439
D.12	Sobrecarga de métodos	441
D.13	Para finalizar	443

E Arrays e ArrayLists 451

E.1	Introdução	452
E.2	Arrays	452
E.3	Declaração e criação de arrays	453
E.4	Exemplos de uso de arrays	455
E.5	Estudo de caso: simulação de embaralhamento e distribuição de cartas	463
E.6	Instrução <code>for</code> melhorada	467
E.7	Passagem de arrays para métodos	468
E.8	Estudo de caso: classe <code>GradeBook</code> usando um array para armazenar as notas	471
E.9	Arrays multidimensionais	476
E.10	Estudo de caso: classe <code>GradeBook</code> usando um array bidimensional	480
E.11	Classe <code>Arrays</code>	485
E.12	Introdução às coleções e à classe <code>ArrayList</code>	488
E.13	Para finalizar	490

F Classes e objetos: uma investigação mais aprofundada 495

F.1	Introdução	496
F.2	Estudo de caso da classe <code>Time</code>	496
F.3	Controle de acesso a membros	500
F.4	Referência aos membros do objeto atual com <code>this</code>	501
F.5	Estudo de caso da classe <code>Time</code> : construtores sobrecarregados	503

F.6	Construtores padrão e sem argumentos	509
F.7	Composição	509
F.8	Enumerações	512
F.9	Coleta de lixo	514
F.10	Membros de classe estáticos	515
F.11	Variáveis de instância final	518
F.12	Pacotes	519
F.13	Acesso ao pacote	520
F.14	Para finalizar	520

G Programação orientada a objetos: herança e polimorfismo **523**

G.1	Introdução à herança	524
G.2	Superclasses e subclasses	525
G.3	Membros protected	526
G.4	Relações entre superclasses e subclasses	527
G.4.1	Criação e uso de uma classe <code>CommissionEmployee</code>	527
G.4.2	Criação e uso de uma classe <code>BasePlusCommissionEmployee</code>	532
G.4.3	Criação de uma hierarquia de herança <code>CommissionEmployee-BasePlusCommissionEmployee</code>	537
G.4.4	Hierarquia de herança <code>CommissionEmployee-BasePlusCommissionEmployee</code> usando variáveis de instância <code>protected</code>	539
G.4.5	Hierarquia de herança <code>CommissionEmployee-BasePlusCommissionEmployee</code> usando variáveis de instância <code>private</code>	542
G.5	Classe <code>Object</code>	547
G.6	Introdução ao polimorfismo	548
G.7	Polimorfismo: um exemplo	549
G.8	Demonstração de comportamento polimórfico	549
G.9	Classes e métodos abstratos	552
G.10	Estudo de caso: sistema de folha de pagamento usando polimorfismo	554
G.10.1	Superclasse abstrata <code>Employee</code>	555
G.10.2	Subclasse concreta <code>SalariedEmployee</code>	558
G.10.3	Subclasse concreta <code>HourlyEmployee</code>	559
G.10.4	Subclasse concreta <code>CommissionEmployee</code>	561
G.10.5	Subclasse concreta indireta <code>BasePlusCommissionEmployee</code>	562
G.10.6	Processamento polimórfico, operador <code>instanceof</code> e <code>downcast</code>	564
G.10.7	Resumo das atribuições permitidas entre variáveis de superclasse e de subclasse	568
G.11	Métodos e classes final	569
G.12	Estudo de caso: criação e uso de interfaces	570
G.12.1	Desenvolvendo uma hierarquia <code>Payable</code>	572
G.12.2	Interface <code>Payable</code>	573
G.12.3	Class <code>Invoice</code>	573
G.12.4	Modificação da classe <code>Employee</code> para implementar a interface <code>Payable</code>	575
G.12.5	Modificação da classe <code>SalariedEmployee</code> para uso na hierarquia <code>Payable</code>	577
G.12.6	Uso da interface <code>Payable</code> para processar objetos <code>Invoice</code> e <code>Employee</code> de forma polimórfica	579

G.13 Interfaces comuns da API Java	580
G.14 Para finalizar	581

H Tratamento de exceções: uma investigação mais aprofundada 586

H.1 Introdução	587
H.2 Exemplo: divisão por zero sem tratamento exceção	587
H.3 Exemplo: tratamento de exceções <code>ArithmeticException</code> e <code>InputMismatchException</code>	589
H.4 Quando usar tratamento de exceção	594
H.5 Hierarquia de exceções da linguagem Java	595
H.6 Bloco <code>finally</code>	597
H.7 Stack unwinding e obtenção de informações de um objeto exceção	601
H.8 Para finalizar	604

I Componentes de interface gráfica do usuário e tratamento de eventos 607

I.1 Introdução	608
I.2 Aparência e comportamento Nimbus	608
I.3 Campos de texto e uma introdução ao tratamento de eventos com classes aninhadas	609
I.4 Tipos de evento comuns em interfaces gráficas e interfaces receptoras	615
I.5 Como funciona o tratamento de eventos	617
I.6 <code>JButton</code>	618
I.7 <code>JComboBox</code> ; uso de uma classe interna anônima para tratamento eventos	622
I.8 Classes adaptadoras	625
I.9 Para finalizar	626

J Outros tópicos da linguagem Java 628

J.1 Introdução	629
J.2 Visão geral das coleções	629
J.3 Classes encapsuladoras de tipos primitivos	630
J.4 Interface <code>Collection</code> e classe <code>Collections</code>	630
J.5 Listas	631
J.5.1 <code>ArrayList</code> e <code>Iterator</code>	631
J.5.2 <code>LinkedList</code>	633
J.5.3 Modos de exibição em coleções e o método <code>asList</code> de <code>Arrays</code>	636
J.6 Métodos de <code>Collections</code>	638
J.6.1 Método <code>sort</code>	638
J.6.2 Método <code>shuffle</code>	640
J.7 Interface <code>Queue</code>	640
J.8 Conjuntos	641
J.9 Mapas	642
J.10 Introdução aos arquivos e fluxos	644
J.11 Classe <code>File</code>	646
J.12 Introdução à serialização de objetos	647

J.13	Introdução ao multithread	648
J.14	Criação e execução de threads com o framework Executor	649
J.15	Visão geral da sincronização de threads	653
J.16	Visão geral das coleções concorrentes	654
J.17	Multithread com interface gráfica do usuário	654
J.18	Para finalizar	660

K Tabela de precedência de operadores **664**

L Tipos primitivos **666**

Índice **667**

Introdução ao Android

1

Objetivos

Neste capítulo, você vai:

- Conhecer a história do Android e do SDK do Android.
- Conhecer o Google Play Store para baixar aplicativos.
- Conhecer os pacotes Android utilizados neste livro para ajudá-lo a criar aplicativos Android.
- Conhecer os conceitos básicos da tecnologia de objetos.
- Conhecer os tipos mais importantes de software para desenvolvimento de aplicativos Android, incluindo o SDK do Android, o SDK do Java, o ambiente de desenvolvimento integrado (IDE) Eclipse e o Android Studio.
- Aprender sobre documentos importantes do Android.
- Testar um aplicativo Android de desenho no Eclipse (no livro impresso) e no Android Studio (online).
- Conhecer as características de excelentes aplicativos Android.



- 1.1 Introdução
- 1.2 Android – o sistema operacional móvel líder mundial
- 1.3 Recursos do Android
- 1.4 Sistema operacional Android
 - 1.4.1 Android 2.2 (Froyo)
 - 1.4.2 Android 2.3 (Gingerbread)
 - 1.4.3 Android 3.0 a 3.2 (Honeycomb)
 - 1.4.4 Android 4.0 a 4.0.4 (Ice Cream Sandwich)
 - 1.4.5 Android 4.1 a 4.3 (Jelly Bean)
 - 1.4.6 Android 4.4 (KitKat)
- 1.5 Baixando aplicativos do Google Play
- 1.6 Pacotes
- 1.7 O SDK do Android
- 1.8 Programação orientada a objetos: uma breve recapitulação
 - 1.8.1 O automóvel como um objeto
 - 1.8.2 Métodos e classes
 - 1.8.3 Instanciação
 - 1.8.4 Reutilização
 - 1.8.5 Mensagens e chamadas de método
 - 1.8.6 Atributos e variáveis de instância
 - 1.8.7 Encapsulamento
 - 1.8.8 Herança
 - 1.8.9 Análise e projeto orientados a objetos
- 1.9 Teste do aplicativo **Doodlz** em um AVD
 - 1.9.1 Executando o aplicativo **Doodlz** no AVD do smartphone Nexus 4
 - 1.9.2 Executando o aplicativo **Doodlz** no AVD de um tablet
 - 1.9.3 Executando o aplicativo **Doodlz** em um aparelho Android
- 1.10 Construção de excelentes aplicativos Android
- 1.11 Recursos para desenvolvimento com Android
- 1.12 Para finalizar

Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios

1.1 Introdução

Bem-vindo ao desenvolvimento de aplicativos Android! Esperamos que o trabalho com este livro seja uma experiência informativa, desafiadora, divertida e recompensadora para você.

Esta parte do livro se destina a *estudantes com experiência em programação com Java*. Utilizamos somente aplicativos funcionais completos; portanto, se você não conhece Java, mas tem experiência com programação orientada a objetos em outra linguagem, como C#, Objective-C/Cocoa ou C++ (com bibliotecas de classe), deve conseguir dominar o material rapidamente, aprendendo Java e programação orientada a objetos no estilo Java à medida que aprende a desenvolver aplicativos Android. Se você não conhece Java, oferecemos também uma introdução ampla e amigável nos apêndices do livro.

Abordagem baseada em aplicativos

O livro usa uma **abordagem baseada em aplicativos** – os novos recursos são discutidos no contexto de aplicativos Android funcionais e completos, com um aplicativo por capítulo. Para cada aplicativo, primeiro o descrevemos e, então, deixamos você *testá-lo*. Em seguida, apresentamos uma breve visão geral das importantes tecnologias **IDE** (ambiente de desenvolvimento integrado) **do Eclipse**, Java e o **SDK** (Software Development Kit) **do Android**, que vamos usar para implementar o aplicativo. Para aplicativos que assim o exigem, conduzimos um acompanhamento *visual* do projeto da interface gráfica do usuário, utilizando o Eclipse. Então, fornecemos a listagem do código-fonte completa, usando números de linha, *sintaxe sombreada* e *realce de código* para enfatizar as partes importantes do código. Mostramos também uma ou mais capturas de tela do aplicativo em execução. Então, fazemos um acompanhamento detalhado do código, enfatizando os novos conceitos de programação introduzidos no aplicativo. Você pode baixar o código-fonte de todos os aplicativos do livro no endereço www.grupoa.com.br.

Para cada capítulo, fornecemos ainda versões do IDE **Android Studio** das instruções específicas do Eclipse. Como o Android Studio é uma versão provisória e está evoluindo rapidamente, fornecemos as respectivas instruções (em inglês) no site do livro

<http://www.deitel.com/books/AndroidHTP2>

Isso nos permitirá manter as instruções atualizadas.

1.2 Android – o sistema operacional móvel líder mundial

As vendas de aparelhos Android estão aumentando rapidamente, criando enormes oportunidades para os desenvolvedores de aplicativos Android.

- A primeira geração de telefones Android foi lançada em outubro de 2008. Em outubro de 2013, um relatório da Strategy Analytics mostrou que o Android tinha 81,3% da fatia de mercado global de *smartphones*, comparados com 13,4% da Apple, 4,1% da Microsoft e 1% do BlackBerry.¹
- De acordo com um relatório do IDC, no final do primeiro trimestre de 2013, o Android tinha 56,5% de participação no mercado global de *tablets*, comparados com 39,6% do iPad da Apple e 3,7% dos tablets Microsoft Windows.²
- Em abril de 2013, mais de 1,5 milhão de aparelhos Android (incluindo smartphones, tablets, etc.) estava sendo ativado diariamente.³
- Quando esta obra estava sendo produzida, havia mais de *um bilhão* de aparelhos Android ativado.⁴
- Atualmente, os dispositivos Android incluem smartphones, tablets, e-readers, robôs, motores a jato, satélites da NASA, consoles de jogos, geladeiras, televisões, câmeras, equipamentos voltados à saúde, relógios inteligentes (smartwatches), sistemas automotivos de “infotainment” de bordo (para controlar rádio, GPS, ligações telefônicas, termostato, etc.) e muitos outros.⁵

1.3 Recursos do Android

Franqueza e código-fonte aberto

Uma vantagem de desenvolver aplicativos Android é a franqueza (ou grau de abertura) da plataforma. O sistema operacional é de *código-fonte aberto* e gratuito. Isso permite ver o código-fonte do Android e como seus recursos são implementados. Você também pode contribuir para o Android relatando erros (consulte <http://source.android.com/source/report-bugs.html>) ou participando nos grupos de discussão do Open Source Project (<http://source.android.com/community/index.html>). Diversos aplicativos An-

¹ <http://blogs.strategyanalytics.com/WSS/post/2013/10/31/Android-Captures-Record-81-Percent-Share-of-Global-Smartphone-Shipments-in-Q3-2013.aspx>.

² <http://www.idc.com/getdoc.jsp?containerId=prUS24093213>.

³ <http://www.technobuffalo.com/2013/04/16/google-daily-android-activations-1-5-million>.

⁴ <http://venturebeat.com/2013/09/03/android-hits-1b-activations-and-will-be-called-kitkat-in-next-version>.

⁵ <http://www.businessweek.com/articles/2013-05-29/behind-the-internet-of-things-is-android-and-its-everywhere>.

droid de código-fonte aberto do Google e outros estão disponíveis na Internet (Fig. 1.1). A Figura 1.2 mostra onde se pode obter o código-fonte do Android, aprender a respeito da filosofia por trás do sistema operacional de código-fonte aberto e obter informações sobre licenciamento.

URL	Descrição
http://en.wikipedia.org/wiki/List_of_open_source_Android_applications	Ampla lista de aplicativos de código-fonte aberto, organizada por categoria (por exemplo, games, comunicação, emuladores, multimídia, segurança).
http://developer.android.com/tools/samples/index.html	Exemplos de aplicativos do Google para a plataforma Android – inclui mais de 60 aplicativos e jogos, como Lunar Lander, Snake e Tic Tac Toe.
http://github.com/	O GitHub permite a você compartilhar seus aplicativos e código-fonte, e colaborar com os projetos de código-fonte aberto de outras pessoas.
http://sourceforge.net	O SourceForge também permite a você compartilhar aplicativos e código-fonte, e colaborar com os projetos de código-fonte aberto de outras pessoas.
http://f-droid.org/	Centenas de aplicativos Android de código-fonte aberto, incluindo o bloqueador de anúncios Adblock Plus, navegação de transporte público aMetro, AnySoftKeyboard (disponível em vários idiomas), o player de música Apollo, o jogo Chinese Checkers, o controlador de peso DroidWeight, Earth Live Wallpaper e muitos mais.
http://blog.interstellr.com/post/39321551640/14-great-android-apps-that-are-also-open-source	Lista 14 aplicativos Android de código-fonte aberto, com links para o código.
http://www.openintents.org/en/libraries	Fornecer quase 100 bibliotecas de código-fonte aberto que podem ser utilizadas para melhorar recursos de aplicativos.
http://www.androidviews.net	Controles de interface gráfica de usuário personalizados para melhorar a aparência de seus aplicativos.
http://www.stackoverflow.com	Stack Overflow é um site de perguntas e respostas para programadores. Os usuários podem votar em cada resposta, e as melhores respostas aparecem no início.

Figura 1.1 Sites de recursos para aplicativos e bibliotecas de código-fonte aberto para Android.

Título	URL
Get Android Source Code	http://source.android.com/source/downloading.html
Governance Philosophy	http://source.android.com/about/philosophy.html
Licenses	http://source.android.com/source/licenses.html
FAQs	http://source.android.com/source/faqs.html

Figura 1.2 Recursos e código-fonte para o sistema operacional Android de código-fonte aberto.

O grau de abertura da plataforma estimula a rápida inovação. Ao contrário do iOS *patenteado* da Apple, que só existe em dispositivos Apple, o Android está disponível em aparelhos de dezenas de fabricantes de equipamento original (OEMs) e em numerosas

operadoras de telecomunicações em todo o mundo. A intensa concorrência entre os OEMs e as operadoras beneficia os consumidores.

Java

Os aplicativos Android são desenvolvidos com Java – uma das linguagens de programação mais usadas do mundo. Essa linguagem foi uma escolha lógica para a plataforma Android, pois é poderosa, gratuita, de código-fonte aberto e milhões de desenvolvedores já a conhecem. Os programadores Java experientes podem se aprofundar rapidamente no desenvolvimento com Android, usando as APIs (interfaces de programação de aplicativo) Android do Google e de outros.

A linguagem Java é orientada a objetos e tem acesso às amplas bibliotecas de classe que ajudam a desenvolver aplicativos poderosos rapidamente. A programação de interfaces gráficas do usuário com Java é *baseada em eventos* – neste livro, você vai escrever aplicativos que respondem a vários *eventos* iniciados pelo usuário, como *toques na tela*. Além de programar partes de seus aplicativos diretamente, você também vai usar os IDEs do Eclipse e do Android Studio para arrastar e soltar convenientemente objetos predefinidos, como botões e caixas de texto para seu lugar na tela, além de rotulá-los e redimensioná-los. Com esses IDEs é possível criar, executar, testar e depurar aplicativos Android de forma rápida e conveniente.

Tela multitouch

Os smartphones Android englobam a funcionalidade de telefone celular, cliente de Internet, MP3 player, console de jogos, câmera digital e muito mais, em um dispositivo portátil com *telas multitouch* coloridas. Com o toque de seus dedos, você pode navegar facilmente entre as opções de usar seu telefone, executar aplicativos, tocar música, navegar na web e muito mais. A tela pode exibir um teclado para digitação de e-mails e mensagens de texto e para a inserção de dados em aplicativos (alguns dispositivos Android também têm teclados físicos).

Gestos

As telas multitouch permitem controlar o aparelho com *gestos* que envolvem apenas um toque ou vários toques simultâneos (Fig. 1.3).

Nome do gesto	Ação física	Utilizado para
Toque rápido (<i>touch</i>)	Tocar rapidamente na tela uma vez.	Abrir um aplicativo, “pressionar” um botão ou um item de menu.
Toque duplo rápido (<i>double touch</i>)	Tocar rapidamente na tela duas vezes.	Ampliar e reduzir imagens, Mapas do Google Maps e páginas web.
Pressionamento longo (<i>long press</i>)	Tocar na tela e manter o dedo na posição.	Selecionar itens em uma visualização – por exemplo, verificar um item em uma lista.
Movimento rápido (<i>swipe</i>)	Tocar e mover rapidamente o dedo na tela, na direção do movimento desejado.	Mover item por item em uma série, como no caso de fotos. Um movimento do swipe para automaticamente no próximo item.
Arrastamento (<i>drag</i>)	Tocar e arrastar o dedo pela tela.	Mover objetos ou ícones, ou rolar precisamente uma página web ou lista.
Zoom de pinça (<i>pinch swipe</i>)	Usando dois dedos, tocar na tela e juntá-los ou afastá-los.	Ampliar e então reduzir a tela (por exemplo, ampliando texto e imagens).

Figura 1.3 Alguns gestos comuns no Android.

Aplicativos incorporados

Os dispositivos Android vêm com vários aplicativos padrão, os quais podem variar de acordo com o aparelho, o fabricante ou o serviço de telefonia móvel. Normalmente, isso inclui **Phone**, **People**, **Email**, **Browser**, **Camera**, **Photos**, **Messaging**, **Calendar**, **Play Store**, **Calculator** e muitos mais.

Web services

Web services são componentes de software armazenados em um computador, os quais podem ser acessados por um aplicativo (ou por outro componente de software) em outro computador por meio da Internet. Com eles, você pode criar **mashups**, os quais permitem desenvolver aplicativos rapidamente, *combinando* web services complementares, frequentemente de diferentes organizações e possivelmente com outras formas de feeds de informação. Por exemplo, o 100 Destinations (www.100destinations.co.uk) combina as fotos e tweets do Twitter com os recursos de mapas do Google Maps, permitindo explorar países em todo o mundo por meio de fotos tiradas por outras pessoas.

O Programmableweb (<http://www.programmableweb.com/>) fornece um catálogo com mais de 9.400 APIs e 7.000 mashups, além de guias práticos e exemplos de código para você criar seus próprios mashups. A Figura 1.4 lista alguns web services populares. De acordo com o Programmableweb, as três APIs mais utilizadas para mashups são: Google Maps, Twitter e YouTube.

Fonte de web services	Como é usada
Google Maps	Serviços de mapas
Twitter	Microblog
YouTube	Busca de vídeo
Facebook	Rede social
Instagram	Compartilhamento de fotos
Foursquare	Check-in móvel
LinkedIn	Rede social para negócios
Groupon	Comércio social
Netflix	Aluguel de filmes
eBay	Leilões pela Internet
Wikipedia	Enciclopédia colaborativa
PayPal	Pagamentos
Last.fm	Rádio na Internet
Amazon eCommerce	Compra de livros e muito mais
Salesforce.com	Gerenciamento de relacionamento com o cliente (CRM)
Skype	Telefonia pela Internet
Microsoft Bing	Busca
Flickr	Compartilhamento de fotos
Zillow	Avaliação de imóveis
Yahoo Search	Busca
WeatherBug	Clima

Figura 1.4 Alguns web services populares (<http://www.programmableweb.com/apis/directory/1?sort=mashups>).

1.4 Sistema operacional Android

O sistema operacional Android foi desenvolvido pela Android, Inc., a qual foi adquirida pelo Google em 2005. Em 2007, foi formada a Open Handset Alliance™ – que agora tem 84 membros (http://www.openhandsetalliance.com/oha_members.html) – para desenvolver, manter e aprimorar o Android, trazendo inovação para a tecnologia móvel, melhorando a experiência do usuário e reduzindo os custos.

Convenção de atribuição de nomes de versão do Android

Cada nova versão do Android recebe um nome de sobremesa, em inglês, em ordem alfabética (Fig. 1.5).

Versão do Android	Nome
Android 1.5	Cupcake
Android 1.6	Donut
Android 2.0 a 2.1	Eclair
Android 2.2	Froyo
Android 2.3	Gingerbread
Android 3.0 a 3.2	Honeycomb
Android 4.0	Ice Cream Sandwich
Android 4.1 a 4.3	Jelly Bean
Android 4.4	KitKat

Figura 1.5 Números de versão do Android e os nomes correspondentes.

1.4.1 Android 2.2 (Froyo)

O **Android 2.2** (também chamado de **Froyo**, lançado em maio de 2010) introduziu o armazenamento externo, permitindo guardar os aplicativos em um dispositivo de memória externo, em vez de apenas na memória interna do aparelho Android. Ele introduziu também o serviço **Android Cloud to Device Messaging (C2DM)**. A **computação em nuvem** (cloud computing) permite utilizar software e dados armazenados na “nuvem” – isto é, acessados em computadores (ou servidores) remotos por meio da Internet e disponíveis de acordo com o pedido – em vez de ficarem armazenados em seu computador de mesa, notebook ou dispositivo móvel. Ela oferece a flexibilidade de aumentar ou diminuir recursos de computação para atender às suas necessidades em dado momento, tornando esse processo mais econômico do que comprar hardware caro para garantir a existência de armazenamento e poder de processamento suficientes para níveis de pico ocasionais. O Android C2DM permite aos desenvolvedores de aplicativos enviarem dados de seus servidores para seus aplicativos instalados em dispositivos Android, mesmo quando os aplicativos *não* estão sendo executados. O servidor avisa os aplicativos para que entrem em contato diretamente com ele para receberem dados atualizados de aplicativo ou do usuário.⁶ Atualmente o C2DM foi substituído pelo Google Cloud Messaging.

⁶ <http://code.google.com/android/c2dm/>.

Para obter mais informações sobre os recursos do Android 2.2 – recursos gráficos OpenGL ES 2.0, framework de mídia e muito mais – visite <http://developer.android.com/about/versions/android-2.2-highlights.html>.

1.4.2 Android 2.3 (Gingerbread)

O **Android 2.3 (Gingerbread)**, lançado no final de 2010, acrescentou mais refinamentos para o usuário, como um teclado redesenhado, recursos de navegação aprimorados, maior eficiência no consumo de energia e muito mais. Adicionou também vários recursos voltados ao desenvolvedor, para comunicação (por exemplo, tecnologias que facilitam fazer e receber ligações dentro de um aplicativo), multimídia (por exemplo, novas APIs de áudio e gráficas) e jogos (por exemplo, melhor desempenho e novos sensores, como um giroscópio para um melhor processamento de movimentos).

Um dos novos recursos mais significativos no Android 2.3 foi o suporte para **comunicação em campo próximo (NFC)** – um padrão de conectividade sem fio de curto alcance que permite a comunicação entre dois dispositivos a uma distância de poucos centímetros. O suporte e os recursos para o NFC variam de acordo com o dispositivo Android. O NFC pode ser usado para pagamentos (por exemplo, um toque de seu aparelho Android habilitado para NFC em um dispositivo de pagamento de uma máquina de refrigerantes), para troca de dados, como contatos e imagens, para emparelhamento de dispositivos e acessórios e muito mais.

Para ver mais recursos voltados ao desenvolvedor de Android 2.3, consulte <http://developer.android.com/about/versions/android-2.3-highlights.html>.

1.4.3 Android 3.0 a 3.2 (Honeycomb)

O **Android 3.0 (Honeycomb)** inclui aprimoramentos na interface do usuário feitos especificamente para dispositivos de tela grande (tais como os tablets), como teclado redesenhado para digitação mais eficiente, interface do usuário em 3D visualmente atraente, navegação mais fácil entre telas dentro de um aplicativo e muito mais. Os novos recursos voltados ao desenvolvedor de Android 3.0 incluíram:

- fragmentos, os quais descrevem partes da interface do usuário de um aplicativo e podem ser combinados em uma única tela ou utilizados em várias telas;
- uma barra de ação persistente na parte superior da tela, fornecendo ao usuário opções para interagir com os aplicativos;
- a capacidade de adicionar layouts de tela grande a aplicativos já existentes, projetados para telas pequenas, a fim de otimizar seu aplicativo para uso em diferentes tamanhos de tela;
- uma interface do usuário visualmente atraente e mais funcional, conhecida como “Holo” por sua aparência e comportamento holográficos;
- um novo framework de animação;
- recursos gráficos e de multimídia aprimorados;
- a capacidade de usar arquiteturas de processador multinúcleo para melhorar o desempenho;
- suporte para Bluetooth ampliado (permitindo, por exemplo, que um aplicativo determine se existem dispositivos conectados, como fones de ouvido ou um teclado); e

- um framework de animação para dar vida a objetos da interface do usuário ou gráficos.

Para ver uma lista dos recursos voltados ao usuário e ao desenvolvedor e tecnologias da plataforma Android 3.0, acesse <http://developer.android.com/about/versions/android-3.0-highlights.html>.

1.4.4 Android 4.0 a 4.0.4 (Ice Cream Sandwich)

O **Android 4.0 (Ice Cream Sandwich)**, lançado em 2011, mesclou o Android 2.3 (Gingerbread) e o Android 3.0 (Honeycomb) em um único sistema operacional para uso em todos os dispositivos Android. Isso permitiu a incorporação, em seus aplicativos para smartphone, de recursos do Honeycomb – como a interface holográfica do usuário, um novo lançador (utilizado para personalizar a tela inicial do dispositivo e para ativar aplicativos) e muito mais – e a fácil adaptação de seus aplicativos para funcionar em diferentes dispositivos. O Ice Cream Sandwich também adicionou várias APIs para uma melhor comunicação entre os dispositivos, acessibilidade para usuários com deficiências (por exemplo, visual), redes sociais e muito mais (Fig. 1.6). Para ver uma lista completa das APIs do Android 4.0, consulte <http://developer.android.com/about/versions/android-4.0.html>.

Recurso	Descrição
Detecção de rosto	Usando a câmera, os dispositivos compatíveis podem determinar a posição dos olhos, nariz e boca do usuário. A câmera também pode monitorar o movimento dos olhos do usuário, permitindo a criação de aplicativos que mudam a perspectiva de acordo com o olhar do usuário.
Operador de câmera virtual	Ao filmar um vídeo com várias pessoas, a câmera focalizará automaticamente a que está falando.
Android Beam	Usando NFC, o Android Beam permite que você encoste dois dispositivos Android para compartilhar conteúdo (como contatos, imagens, vídeos).
Wi-Fi Direct	As APIs Wi-Fi P2P (peer-to-peer) permitem conectar vários aparelhos Android utilizando Wi-Fi. Sem utilizar fios, eles podem se comunicar a uma distância maior do que usando Bluetooth.
Social API	Acesse e compartilhe informações de contato entre redes sociais e aplicativos (com a permissão do usuário).
Calendar API	Adicione e compartilhe eventos entre vários aplicativos, gerencie alertas e participantes e muito mais.
APIs de acessibilidade	Use as novas APIs Accessibility Text-to-Speech para melhorar a experiência do usuário em seus aplicativos para pessoas com deficiências, como deficientes visuais, e muito mais. O modo explorar por toque permite aos usuários deficientes visuais tocar em qualquer parte na tela e ouvir uma voz descrevendo o conteúdo tocado.
Framework Android@Home	Use o framework Android@Home para criar aplicativos que controlam utensílios nas casas dos usuários, como termostatos, sistemas de irrigação, lâmpadas elétricas em rede e muito mais.
Dispositivos Bluetooth voltados à saúde	Crie aplicativos que se comunicam com dispositivos Bluetooth voltados à saúde, como balanças, monitores de batimento cardíaco e muito mais.

Figura 1.6 Alguns recursos para desenvolvedores de Android Ice Cream Sandwich (<http://developer.android.com/about/versions/android-4.0.html>).

1.4.5 Android 4.1 a 4.3 (Jelly Bean)

O **Android Jelly Bean**, lançado em 2012, inclui suporte para telas de vídeo externas, segurança aprimorada, melhorias na aparência (por exemplo, widgets de aplicativo que podem ser dimensionados e notificações de aplicativo maiores) e no desempenho, que aperfeiçoam a troca entre aplicativos e telas (Fig. 1.7). Para ver a lista de recursos do Jelly Bean, consulte <http://developer.android.com/about/versions/jelly-bean.html>

Recurso	Descrição
Android Beam	O Android Beam pode ser usado para unir facilmente seu smartphone ou tablet a alto-falantes Bluetooth® sem fio ou fones de ouvido especiais.
Widgets de tela de bloqueio	Crie widgets que aparecem na tela do usuário quando o aparelho está bloqueado ou modifique seus widgets de tela inicial existentes para que também fiquem visíveis nessa situação.
Photo Sphere	APIs para trabalhar com os novos recursos de foto panorâmica. Permitem aos usuários tirar fotografias em 360° semelhantes às usadas no Street View do Google Maps.
Daydreams	Daydreams são protetores de tela interativos, ativados quando um aparelho está encaixado ou carregando. Eles podem reproduzir áudio e vídeo e responder às interações do usuário.
Suporte para idiomas	Novos recursos ajudam seus aplicativos a atingir usuários internacionais, tais como texto bidirecional (da esquerda para a direita ou da direita para a esquerda), teclados internacionais, layouts de teclado adicionais e muito mais.
Opções para o desenvolvedor	Vários novos recursos de monitoramento e depuração ajudam a melhorar seus aplicativos, como os relatórios de erros, que incluem uma captura de tela e informações de estado do dispositivo.

Figura 1.7 Alguns recursos do Android Jelly Bean (<http://developer.android.com/about/versions/jelly-bean.html>).

1.4.6 Android 4.4 (KitKat)

O **Android 4.4 KitKat**, lançado em outubro de 2013, inclui várias melhorias de desempenho que tornam possível executar o sistema operacional em todos os aparelhos Android, inclusive em dispositivos mais velhos com restrição de memória, os quais são particularmente populares nos países em desenvolvimento.⁷

Permitir que mais usuários atualizem para o KitKat reduzirá a “fragmentação” de versões de Android no mercado, o que tem sido um desafio para os desenvolvedores que antes tinham de projetar aplicativos para serem executados em várias versões do sistema operacional ou limitar seu mercado em potencial, tendo como alvo aplicativos para uma versão específica.

O Android KitKat inclui também aprimoramentos na segurança e na acessibilidade, recursos gráficos e de multimídia melhorados, ferramentas de análise de uso da memória e muito mais. A Figura 1.8 lista alguns recursos novos importantes do KitKat. Para ver uma lista completa, consulte

<http://developer.android.com/about/versions/kitkat.html>

⁷ <http://techcrunch.com/2013/10/31/android-4-4-kitkat-google/>.

Recurso	Descrição
Modo imersivo	A barra de status na parte superior da tela e os botões de menu na parte inferior podem ser ocultos, permitindo que seus aplicativos ocupem uma parte maior da tela. Os usuários podem acessar a barra de status fazendo um pressionamento forte (swipe) de cima para baixo na tela, e a barra de sistema (com os botões voltar, iniciar e aplicativos recentes) fazendo um pressionamento forte de baixo para cima.
Framework de impressão	Crie funcionalidade de impressão em seus aplicativos, incluindo localizar impressoras disponíveis via Wi-Fi ou na nuvem, selecionar o tamanho do papel e especificar as páginas a serem impressas.
Framework de acesso ao armazenamento	Crie provedores de armazenamento de documentos que permitam aos usuários localizar, criar e editar arquivos (como documentos e imagens) entre vários aplicativos.
Provedor de SMS	Crie aplicativos de SMS (Short Message Service) ou MMS (Multimedia Messaging Service) com o novo provedor de SMS e as novas APIs. Agora os usuários podem selecionar seus aplicativos de troca de mensagens padrão.
Framework de transições	O novo framework torna mais fácil criar animações de transição.
Gravação de tela	Grave um vídeo de seu aplicativo em ação para criar tutoriais e materiais de marketing.
Acessibilidade melhorada	A API gerenciadora de títulos permite aos aplicativos verificar as preferências de títulos do usuário (por exemplo, idioma, estilos de texto e muito mais).
Chromium WebView	Dê suporte aos padrões mais recentes para exibir conteúdo web, incluindo HTML5, CSS3 e uma versão mais rápida de JavaScript.
Detector e contador de passos	Crie aplicativos que identificam se o usuário está correndo, andando ou subindo a escada e que contam o número de passos.
Host Card Emulator (HCE)	O HCE permite que qualquer aplicativo realize transações NFC seguras (como pagamentos móveis) sem a necessidade de um elemento de segurança no cartão SIM, controlado pela operadora de telefonia celular.

Figura 1.8 Alguns recursos do Android KitKat (<http://developer.android.com/about/versions/kitkat.html>).

1.5 Baixando aplicativos do Google Play

Quando este livro estava sendo produzido, havia mais de 1 milhão de aplicativos no **Google Play**, e o número está crescendo rapidamente.⁸ A Figura 1.9 lista alguns aplicativos populares gratuitos e pagos. Você pode baixar aplicativos por meio do aplicativo **Play Store** instalado no dispositivo. Pode também conectar-se em sua conta no Google Play, no endereço <http://play.google.com>, com seu navegador web e, então, especificar o dispositivo Android no qual deseja instalar o aplicativo. Ele será baixado por meio da conexão Wi-Fi ou 3G/ 4G do aparelho. No Capítulo 9, intitulado “Google Play e questões de comercialização de aplicativos”, discutimos sobre mais lojas de aplicativos, sobre como oferecer seus aplicativos gratuitamente ou cobrando uma taxa, sobre o preço dos aplicativos e muito mais.

⁸ en.wikipedia.org/wiki/Google_Play.

Categoria do Google Play	Alguns aplicativos populares na categoria
Clima	WeatherBug, AccuWeather, The Weather Channel
Compras	eBay, Amazon Mobile, Groupon, The Coupons App
Comunicação	Facebook Messenger, Skype™, GrooVe IP
Cuidados médicos	Epocrates, ICE: In Case of Emergency, Medscape®
Educação	Duolingo: Learn Languages Free, TED, Mobile Observatory
Empresas	Office Suite Pro 7, Job Search, Square Register, GoToMeeting
Entretenimento	SketchBook Mobile, Netflix, Fandango® Movies, iFunny :)
Esportes	SportsCenter for Android, NFL '13, Team Stream™
Estilo de vida	Zillow Real Estate, Epicurious Recipe App, Family Locator
Finanças	Mint.com Personal Finance, Google Wallet, PayPal
Fotografia	Camera ZOOM FX, Photo Grid, InstaPicFrame for Instagram
Humor	ComicRack, Memedroid Pro, Marvel Comics, Comic Strips
Jogos: Arcade & ação	Minecraft–Pocket Edition, Fruit Ninja, Angry Birds
Jogos: Cartas & cassino	Solitaire, Slots Delux, UNO™ & Friends, DH Texas Poker
Jogos: Casual	Candy Crush Saga, Hardest Game Ever 2, Game Dev Story
Jogos: Quebra-cabeças	Where's My Water?, Draw Something, Can You Escape
Livros e referências	Kindle, Wikipedia, Audible for Android, Google Play Books
Multimídia & vídeo	MX Player, YouTube, KeepSafe Vault, RealPlayer®
Música & áudio	Pandora®, Shazam, Spotify, Ultimate Guitar Tabs & Chords
Notícias & revistas	Flipboard, Pulse News, CNN, Engadget, Dripler
Personalização	Beautiful Widgets Pro, Zedge™, GO Launcher EX
Plano de fundo interativo	PicsArt, GO Launcher EX, Beautiful Widgets Pro
Produtividade	Adobe® Reader®, Dropbox, Google Keep, SwiftKey Keyboard
Saúde e cond. físico	RunKeeper, Calorie Counter, Workout Trainer, WebMD®
Social	Facebook®, Instagram, Vine, Twitter, Snapchat, Pinterest
Transporte	Uber, Trapster, Lyft, Hailo™, Ulysse Speedometer
Utilitários	Titanium Backup PRO, Google Translate, Tiny Flashlight®
Viagens & local	Waze, GasBuddy, KAYAK, TripAdvisor, OpenTable®
Widgets	Zillow, DailyHoroscope, Starbucks, Family Locator

Figura 1.9 Alguns aplicativos Android populares no Google Play.

1.6 Pacotes

O Android usa um conjunto de *pacotes*, que são grupos nomeados de classes predefinidas e relacionadas. Alguns pacotes são específicos do Android, outros são do Java e do Google. Esses pacotes permitem acessar convenientemente os recursos do sistema operacional Android e incorporá-los em seus aplicativos. Os pacotes do Android ajudam a criar aplicativos que obedecem às convenções de aparência e comportamento e diretrizes de estilo exclusivas do Android (<http://developer.android.com/design/index.html>). A Figura 1.10 lista os pacotes discutidos neste livro. Para uma lista completa de pacotes Android, consulte developer.android.com/reference/packages.html.

Pacote	Descrição
<code>android.app</code>	Inclui classes de alto nível do modelo de aplicativos Android. (Aplicativo Tip Calculator do Capítulo 3.)
<code>android.content</code>	Acesso e publicação de dados em um dispositivo. (Aplicativo Cannon Game do Capítulo 6.)
<code>android.content.res</code>	Classes para acessar recursos de aplicativo (por exemplo, mídia, cores, desenhos, etc.) e informações de configuração de dispositivo que afetam o comportamento dos aplicativos. (Aplicativo Flag Quiz do Capítulo 5.)
<code>android.database</code>	Manipulação de dados retornados pelo provedor de conteúdo. (Aplicativo Address Book do Capítulo 8.)
<code>android.database.sqlite</code>	Gerenciamento de banco de dados SQLite para bancos de dados privados. (Aplicativo Address Book do Capítulo 8.)
<code>android.graphics</code>	Ferramentas gráficas usadas para desenhar na tela. (Aplicativos Flag Quiz do Capítulo 5 e Doodlz do Capítulo 7.)
<code>android.hardware</code>	Suporte para hardware de dispositivo. (Aplicativo Doodlz do Capítulo 7.)
<code>android.media</code>	Classes para manipular interfaces de áudio e vídeo. (Aplicativo Cannon Game do Capítulo 6.)
<code>android.net</code>	Classes de acesso à rede. (Aplicativo Twitter® Searches do Capítulo 4.)
<code>android.os</code>	Serviços de sistemas operacionais. (Aplicativo Tip Calculator do Capítulo 3.)
<code>android.preference</code>	Trabalho com as preferências do usuário de um aplicativo. (Aplicativo Flag Quiz do Capítulo 5.)
<code>android.provider</code>	Acesso a provedores de conteúdo Android. (Aplicativo Doodlz do Capítulo 7.)
<code>android.support.v4.print</code>	Recursos da Android Support Library para usar o framework de impressão do Android 4.4. (Aplicativo Doodlz do Capítulo 7.)
<code>android.text</code>	Renderização e monitoramento de texto no dispositivo. (Aplicativo Tip Calculator do Capítulo 3.)
<code>android.util</code>	Métodos utilitários e utilitários XML. (Aplicativo Cannon Game do Capítulo 6.)
<code>android.widget</code>	Classes de interface do usuário para widgets. (Aplicativo Tip Calculator do Capítulo 3.)
<code>android.view</code>	Classes de interface do usuário para layout e interações do usuário. (Aplicativo Twitter® Searches do Capítulo 4.)
<code>java.io</code>	Streaming, serialização e acesso ao sistema de arquivo de recursos de entrada e saída. (Aplicativo Flag Quiz do Capítulo 5.)
<code>java.text</code>	Classes de formatação de texto. (Aplicativo Twitter® Searches do Capítulo 4.)
<code>java.util</code>	Classes utilitárias. (Aplicativo Twitter® Searches do Capítulo 4.)
<code>android.graphics.drawable</code>	Classes para elementos somente de exibição (como gradientes, etc.). (Aplicativo Flag Quiz do Capítulo 5.)

Figura 1.10 Pacotes Android e Java utilizados neste livro, listados com o capítulo em que aparecem pela primeira vez.

1.7 O SDK do Android

O SDK (Software Development Kit) do Android fornece as ferramentas necessárias para construir aplicativos Android. Ele está disponível gratuitamente no site Android Developers. Consulte a seção “Antes de começar” para ver os detalhes completos sobre como baixar as ferramentas necessárias para desenvolver aplicativos Android, incluindo o Java SE, o Android SDK/ADT Bundle (o qual inclui o IDE Eclipse) e o IDE Android Studio.

Android SDK/ADT Bundle

O Android SDK/ADT Bundle – que inclui o IDE Eclipse – é o ambiente de desenvolvimento integrado mais amplamente usado para desenvolvimento com Android. Alguns

desenvolvedores utilizam somente um editor de texto e ferramentas de linha de comando para criar aplicativos Android. O IDE Eclipse inclui:

- Editor de código com suporte para sintaxe colorida e numeração de linha
- Recuo (*auto-indenting*) e preenchimento automáticos (isto é, sugestão de tipo)
- Depurador
- Sistema de controle de versão
- Suporte para refatoração

Você vai usar o Eclipse na Seção 1.9 para testar o aplicativo **Doodlz**. A partir do Capítulo 2, intitulado “Aplicativo **Welcome**”, você vai usar o Eclipse para construir aplicativos.

Android Studio

O **Android Studio**, um novo IDE Java Android baseado no IDE JetBrains IntelliJ IDEA (<http://www.jetbrains.com/idea/>), foi anunciado em 2013 e é o IDE Android do futuro preferido do Google. Quando este livro estava sendo produzido, o Android Studio só estava disponível como *apresentação prévia de teste* – muitos de seus recursos ainda estavam em desenvolvimento. Para cada capítulo, fornecemos também versões do Android Studio (em inglês) das instruções específicas do Eclipse no site do livro

<http://www.deitel.com/books/AndroidHTP2>

Para saber mais sobre o Android Studio, como instalar e migrar do Eclipse, visite <http://developer.android.com/sdk/installing/studio.html>.

Plugin ADT para Eclipse

O **Plugin ADT (Android Development Tools) para Eclipse** (parte do Android SDK/ADT Bundle) permite criar, executar e depurar aplicativos Android, exportá-los para distribuição (por exemplo, carregá-los no Google Play) e muito mais. O ADT também contém uma ferramenta de projeto visual de interface gráfica do usuário. Os componentes da interface gráfica do usuário podem ser arrastados e soltos no lugar para formar interfaces, sem codificação. Você vai aprender mais sobre o ADT no Capítulo 2.

O emulador do Android

O **emulador do Android**, incluído no SDK do Android, permite executar aplicativos Android em um ambiente simulado dentro do Windows, Mac OS X ou Linux, sem usar um dispositivo Android real. O emulador exibe uma janela de interface de usuário realista. Ele será particularmente útil se você não tiver acesso a dispositivos Android para teste. Obviamente, você deve testar seus aplicativos em diversos dispositivos Android antes de carregá-los no Google Play.

Antes de executar um aplicativo no emulador, você precisa criar um **AVD (Android Virtual Device ou Dispositivo Android Virtual)**, o qual define as características do dispositivo em que o teste vai ser feito, incluindo o hardware, a imagem do sistema, o tamanho da tela, o armazenamento de dados e muito mais. Se quiser testar seus aplicativos para vários dispositivos Android, você precisará criar AVDs separados para emular cada equipamento exclusivo ou usar um serviço (como testdroid.com ou appthwack.com) que permita testar em muitos dispositivos diferentes.

Usamos o emulador (não um dispositivo Android real) para obter a maioria das capturas de tela do Android deste livro. No emulador, você pode reproduzir a maioria dos gestos (Fig. 1.11) e controles (Fig. 1.12) do Android usando o teclado e o mouse

de seu computador. Os gestos com os dedos no emulador são um pouco limitados, pois seu computador provavelmente não consegue simular todos os recursos de hardware do Android. Por exemplo, para testar aplicativos de GPS no emulador, você precisa criar arquivos que simulem leituras em um GPS. Além disso, embora seja possível simular mudanças de orientação (para o modo *retrato* ou *paisagem*), simular leituras de **acelerômetro** em particular (o acelerômetro permite que o dispositivo responda à aceleração para cima/para baixo, para esquerda/para direita e para frente/para trás) exige recursos que não estão presentes no emulador. Há um *Sensor Simulator* disponível em

<https://code.google.com/p/openintents/wiki/SensorSimulator>

que pode ser usado para enviar informações de sensor simuladas para um AVD a fim de testar outros recursos de sensor em seus aplicativos. A Figura 1.13 lista funcionalidades do Android que *não* estão disponíveis no emulador. Contudo, você pode carregar seu aplicativo (fazer upload) em um dispositivo Android para testar esses recursos. Você vai começar a criar AVDs e a usar o emulador para desenvolver aplicativos Android no aplicativo **Welcome** do Capítulo 2.

Gesto	Ação do emulador
Toque rápido (<i>touch</i>)	Clicar com o mouse uma vez. Apresentado no aplicativo Tip Calculator do Capítulo 3.
Duplo toque rápido (<i>double touch</i>)	Clicar duas vezes com o mouse. Apresentado no aplicativo Cannon Game do Capítulo 6.
Pressionamento longo (<i>long press</i>)	Clicar e manter o botão do mouse pressionado.
Arrastamento (<i>drag</i>)	Clicar, manter o botão do mouse pressionado e arrastar. Apresentado no aplicativo Cannon Game do Capítulo 6.
Pressionamento forte (<i>swipe</i>)	Clicar e manter o botão do mouse pressionado, mover o cursor na direção do pressionamento e soltar o mouse. Apresentado no aplicativo Address Book do Capítulo 8.
Zoom de pinça (<i>pinch zoom</i>)	Manter a tecla <i>Ctrl</i> (<i>Control</i>) pressionada. Vão aparecer dois círculos que simulam os dois toques. Mova os círculos para a posição inicial, clique e mantenha o botão do mouse pressionado e arraste os círculos até a posição final.

Figura 1.11 Gestos do Android no emulador.

Controle	Ação do emulador
Back (voltar)	<i>Esc</i>
Botão call/dial (chamar/discar)	<i>F3</i>
Camera (câmera)	<i>Ctrl-KEYPAD_5</i> , <i>Ctrl-F3</i>
Botão End Call (finalizar chamada)	<i>F4</i>
Home (início)	<i>Botão Home</i>
Menu (tecla programável esquerda)	<i>F2</i> ou botão <i>Page Up</i>
Botão Power (ligar/desligar)	<i>F7</i>
Search (pesquisar)	<i>F5</i>
* (tecla programável direita)	<i>Shift-F2</i> ou botão <i>Page Down</i>
Girar para a orientação anterior	<i>KEYPAD_7</i> , <i>Ctrl-F11</i>
Girar para a próxima orientação	<i>KEYPAD_9</i> , <i>Ctrl-F12</i>

Figura 1.12 Controles de hardware do Android no emulador (para ver mais controles, acesse <http://developer.android.com/tools/help/emulator.html>). (continua)

Controle	Ação do emulador
Ativar/desativar rede celular	F8
Botão Volume Up (aumentar volume)	KEYPAD_PLUS, Ctrl-F5
Botão Volume Down (diminuir volume)	KEYPAD_MINUS, Ctrl-F6

Figura 1.12 Controles de hardware do Android no emulador (para ver mais controles, acesse <http://developer.android.com/tools/help/emulator.html>).

Funcionalidades do Android não disponíveis no emulador
• Fazer ou receber ligações telefônicas reais (o emulador só permite chamadas simuladas) • Bluetooth • Conexões USB • Fones de ouvido ligados ao dispositivo • Determinar estado de conexão do telefone • Determinar a carga da bateria ou o estado de carga de energia • Determinar a inserção/ejeção de cartão SD • Sensores (acelerômetro, barômetro, bússola, sensor de luz, sensor de proximidade)

Figura 1.13 Funcionalidades do Android não disponíveis no emulador (<http://developer.android.com/tools/devices/emulator.html>).

1.8 Programação orientada a objetos: uma breve recapitulação

O Android utiliza técnicas de programação orientada a objetos. Portanto, nesta seção vamos rever os fundamentos da tecnologia de objetos. Usamos todos esses conceitos neste livro.

Construir software de forma rápida, correta e econômica continua sendo um objetivo ilusório em uma época em que a demanda por software novo e mais poderoso está aumentando. Os *objetos* ou, mais precisamente, as *classes* de onde os objetos vêm, são basicamente componentes de software *reutilizáveis*. Existem objetos data, objetos tempo, objetos áudio, objetos vídeo, objetos automóveis, objetos pessoas, etc. Praticamente qualquer *substantivo* pode ser representado de forma razoável como um objeto de software, em termos de *atributos* (por exemplo, nome, cor e tamanho) e *comportamentos* (por exemplo, cálculo, movimento e comunicação). Os desenvolvedores de software estão descobrindo que usar uma estratégia de projeto e implementação modular e orientada a objetos pode tornar os grupos de desenvolvimento de software muito mais produtivos do que poderiam ser com técnicas anteriormente populares, como a “programação estruturada” – frequentemente, os programas orientados a objetos são mais fáceis de entender, corrigir e modificar.

1.8.1 O automóvel como um objeto

Para ajudá-lo a entender os objetos e seu conteúdo, vamos começar com uma analogia simples. Suponha que você queira *dirigir um carro e fazê-lo ir mais rápido pressionando o pedal do acelerador*. O que precisa acontecer antes que você possa fazer isso? Bem, antes que você possa dirigir um carro, alguém tem de *projetá-lo*. Normalmente, um carro começa com desenhos de engenharia, semelhantes às *plantas baixas* que descrevem o

projeto de uma casa. Esses desenhos incluem o projeto de um pedal de acelerador. O pedal *esconde* do motorista os mecanismos complexos que fazem o carro ir mais rápido, assim como o pedal do freio *esconde* os mecanismos que diminuem a velocidade do carro e o volante *esconde* os mecanismos que fazem o carro desviar. Isso permite que pessoas com pouco ou nenhum conhecimento do funcionamento de motores, freios e mecanismos de direção dirijam um carro facilmente. Assim como não se pode fazer comida na cozinha de uma planta baixa, tampouco é possível dirigir os desenhos de engenharia de um carro. Antes que você possa dirigir um carro, ele precisa ser *construído* a partir dos desenhos de engenharia que o descrevem. Um carro pronto tem um pedal de acelerador *real* para fazê-lo ir mais rápido, mas mesmo isso não é suficiente – o carro não acelera sozinho (espera-se!), de modo que o motorista precisa *pressionar* o pedal para acelerá-lo.

1.8.2 Métodos e classes

Usemos nosso exemplo do carro para introduzir alguns conceitos importantes de programação orientada a objetos. Executar uma tarefa em um programa exige um **método**. O método contém as instruções do programa que realmente executam suas tarefas. O método oculta essas instruções de seu usuário, assim como o pedal do acelerador de um carro oculta do motorista os mecanismos que fazem o carro ir mais rápido. Uma unidade de programa, chamada **classe**, contém os métodos que executam as tarefas da classe. Por exemplo, uma classe que represente uma conta bancária poderia conter um método para *depositar* dinheiro em uma conta, outro para *sacar* dinheiro de uma conta e um terceiro para *informar* o saldo da conta. Uma classe é conceitualmente semelhante aos desenhos de engenharia de um carro, os quais contêm o projeto de um pedal de acelerador, de um volante, etc.

1.8.3 Instanciação

Assim como alguém precisa *construir um carro* a partir de seus desenhos de engenharia antes que você possa dirigi-lo, é preciso *construir um objeto* de uma classe antes que um programa possa executar as tarefas que os métodos da classe definem. O processo de fazer isso é chamado de *instanciação*. Um objeto, então, é uma **instância** de sua classe.

1.8.4 Reutilização

Assim como os desenhos de engenharia de um carro podem ser *reutilizados* muitas vezes para construir muitos carros, você pode *reutilizar* uma classe muitas vezes para construir muitos objetos. A **reutilização** de classes já existentes ao construir novas classes e programas economiza tempo e trabalho. A reutilização também o ajuda a construir sistemas mais confiáveis e eficientes, pois as classes e componentes já existentes frequentemente passaram por extensivos *testes*, *depuração* e otimização de *desempenho*. Assim como a noção de *partes intercambiáveis* foi fundamental para a Revolução Industrial, as classes reutilizáveis são fundamentais para a revolução na área de software estimulada pela tecnologia de objetos.

1.8.5 Mensagens e chamadas de método

Quando você dirige um carro, pressionar o acelerador envia uma *mensagem* para o carro executar uma tarefa – ou seja, ir mais rápido. Da mesma forma, você *envia mensagens para um objeto*. Cada mensagem é uma **chamada de método** que diz a um método do objeto para que execute sua tarefa. Por exemplo, um programa poderia chamar um método *depositar* de um objeto de conta bancária em particular para que ele aumentasse o saldo da conta.

1.8.6 Atributos e variáveis de instância

Um carro, além de ter recursos para cumprir tarefas, também tem *atributos*, como cor, número de portas, capacidade de combustível no tanque, velocidade atual e registro do total de quilômetros rodados (isto é, a leitura de seu odômetro). Assim como seus recursos, os atributos do carro são representados como parte de seu projeto nos diagramas de engenharia (os quais, por exemplo, incluem um odômetro e um medidor de combustível). Quando você dirige um carro, esses atributos são transportados junto com o veículo. Todo carro mantém seus *próprios* atributos. Por exemplo, cada carro sabe a quantidade de combustível existente no tanque, mas *não* o quanto existe nos tanques de *outros* carros.

Da mesma forma, um objeto tem atributos que carrega consigo quando usado em um programa. Esses atributos são especificados como parte da classe do objeto. Por exemplo, um objeto conta bancária tem um *atributo saldo* que representa a quantidade de dinheiro existente na conta. Cada objeto conta bancária sabe o saldo da conta que representa, mas *não* os saldos das *outras* contas bancárias. Os atributos são especificados pelas **variáveis de instância** da classe.

1.8.7 Encapsulamento

As classes **encapsulam** (isto é, empacotam) atributos e métodos nos objetos – os atributos e métodos de um objeto estão intimamente relacionados. Os objetos podem se comunicar entre si, mas normalmente não podem saber como outros objetos são implementados – os detalhes da implementação ficam *ocultos* dentro dos próprios objetos. Essa **ocultação de informações** é fundamental para a boa engenharia de software.

1.8.8 Herança

Uma nova classe de objetos pode ser criada rápida e convenientemente por meio de **herança** – a nova classe absorve as características de outra já existente, possivelmente personalizando-as e adicionando suas próprias características exclusivas. Em nossa analogia automobilística, um “conversível” certamente *é um* objeto da classe mais *geral* “automóvel”, mas mais *especificamente* o teto pode ser levantado ou abaixado.

1.8.9 Análise e projeto orientados a objetos

Como você vai criar o código de seus programas? Talvez, como muitos programadores, você simplesmente ligue o computador e comece a digitar. Essa estratégia pode funcionar para programas pequenos, mas e se você fosse solicitado a criar um sistema de software para controlar milhares de caixas eletrônicos para um grande banco? Ou então suponha que você fosse solicitado a trabalhar em uma equipe com mil desenvolvedores de software para construir o próximo sistema de controle de tráfego aéreo dos Estados Unidos. Para projetos tão grandes e complexos, você não deve simplesmente sentar e começar a escrever programas.

Para criar as melhores soluções, você deve seguir um processo de **análise** detalhado para determinar os **requisitos** de seu projeto (isto é, definir *o que* o sistema deve fazer) e desenvolver um **projeto** que os satisfaça (isto é, decidir *como* o sistema deve fazer isso). De maneira ideal, você passaria por esse processo e examinaria o projeto cuidadosamente (e teria seu projeto examinado por outros profissionais de software) antes de escrever qualquer código. Se esse processo envolve analisar e projetar seu sistema do ponto de vista orientado a objetos, ele é denominado **processo de análise e projeto orientados a objetos** (OOAD – Object-Oriented Analysis and Design). Linguagens como Java são orientadas a objetos. Programar em uma linguagem assim, o que é chamado de

programação orientada a objetos (OOP), permite implementar um projeto orientado a objetos como um sistema funcional.

1.9 Teste do aplicativo Doodlz em um AVD

Nesta seção, você vai executar e interagir com seu primeiro aplicativo Android. O aplicativo **Doodlz** permite que você “pinte” na tela arrastando os dedos. É possível controlar o tamanho do pincel e as cores usando as escolhas fornecidas no *menu de opções* do aplicativo. Não há necessidade de examinar o código do aplicativo – você vai construí-lo e estudar seu código no Capítulo 7. Os passos a seguir mostram como importar o projeto do aplicativo para o Eclipse e como testar o aplicativo no AVD (Dispositivo Android Virtual) do Nexus 4 que você configurou na seção “Antes de começar”, após o Prefácio. Posteriormente nesta seção, vamos discutir também como se executa o aplicativo no AVD de um tablet e em um dispositivo Android. Quando o aplicativo é executado em um AVD, você pode criar uma nova pintura “arrastando seu dedo” em qualquer lugar na tela de desenho. Você “toca” na tela usando o mouse.

IDEs Android SDK/ADT Bundle e Android Studio

As capturas de tela de IDE nos passos a seguir (e por todo este livro) foram obtidas em um computador com Windows 7, Java SE 7, JDK e Android SDK/ADT Bundle que você instalou na seção “Antes de começar”. Como o Android Studio é uma versão provisória e está evoluindo rapidamente, fornecemos as respectivas instruções (em inglês) para este teste no site do livro

www.deitel.com/books/AndroidHTP2.

Isso nos permitirá atualizar as instruções em resposta às alterações do Google. Tanto o Android SDK/ADT Bundle como o Android Studio usam o *mesmo* emulador do Android; portanto, quando um aplicativo estiver executando em um AVD, os passos serão idênticos.

1.9.1 Executando o aplicativo Doodlz no AVD do smartphone Nexus 4

Para testar o aplicativo **Doodlz**, execute os passos a seguir:

1. **Verificando sua configuração.** Caso ainda não tenha feito isso, execute os passos especificados na seção “Antes de começar”, localizada após o Prefácio.
2. **Abrindo o Eclipse.** Abra a subpasta `eclipse` da pasta de instalação do Android SDK/ADT Bundle e, em seguida, clique duas vezes no ícone do Eclipse (📁 ou 📄, dependendo de sua plataforma).
3. **Especificando o local de sua área de trabalho.** Quando a janela **Workspace Launcher** aparecer, especifique onde gostaria de armazenar os aplicativos que você vai criar e, em seguida, clique em **OK**. Utilizamos o local padrão – uma pasta chamada `workspace` em seu diretório de usuário. Uma **área de trabalho** é uma coleção de projetos, e cada projeto em geral é um aplicativo ou uma biblioteca que pode ser compartilhada entre aplicativos. Cada área de trabalho também tem suas próprias configurações, como o local de exibição das várias subjanelas do Eclipse. É possível ter muitas áreas de trabalho e trocar entre elas para diferentes tarefas de desenvolvimento – por exemplo, você poderia ter áreas de trabalho separadas para desenvolvimento de aplicativos Android, aplicativos Java e aplicativos web, cada uma com

suas configurações personalizadas. Se essa é a primeira vez que você abre o Eclipse, a página **Welcome** (Fig. 1.14) é exibida.



Figura 1.14 Página **Welcome** no Eclipse.

4. **Ativando o AVD do Nexus 4.** Para este teste, vamos usar o AVD do smartphone Nexus 4 que você configurou para Android 4.4 (KitKat) na seção “Antes de começar” – na Seção 1.9.2, mostraremos o aplicativo em execução em um AVD de tablet. Um AVD pode demorar vários minutos para carregar; portanto, você deve ativá-lo com antecedência e mantê-lo executando em segundo plano, enquanto está construindo e testando seus aplicativos. Para ativar o AVD do Nexus 4, selecione **Window > Android Virtual Device Manager** a fim de exibir a caixa de diálogo **Android Virtual Device Manager** (Fig. 1.15). Selecione o AVD do Nexus 4 para Android KitKat e clique em **Start...**; em seguida, clique no botão **Launch** na caixa de diálogo **Launch Options** que aparece. Não tente executar o aplicativo antes que o AVD acabe de ser carregado. Quando o AVD aparecer como mostrado na Fig. 1.16, desbloqueie-o arrastando o cursor do mouse a partir do ícone de cadeado para a margem da tela.

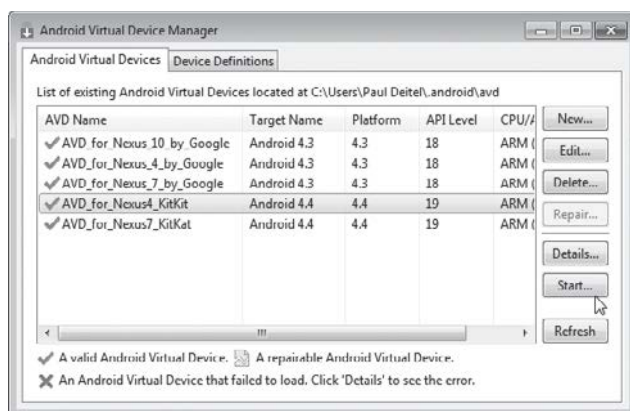


Figura 1.15 Caixa de diálogo **Android Virtual Device Manager**.



Arraste o cursor do mouse a partir do ícone de cadeado para a margem da tela a fim de desbloquear o AVD

Figura 1.16 Tela inicial do AVD do Nexus 4 (para Android 4.4) quando ele termina de carregar.

5. **Importando o projeto do aplicativo Doodlz.** Selecione **File > Import...** para abrir a caixa de diálogo **Import** (Fig. 1.17(a)). Expanda o nó **General** e selecione **Existing Projects into Workspace**; em seguida, clique em **Next >** para ir ao passo **Import Pro-**

a) Caixa de diálogo **Import**

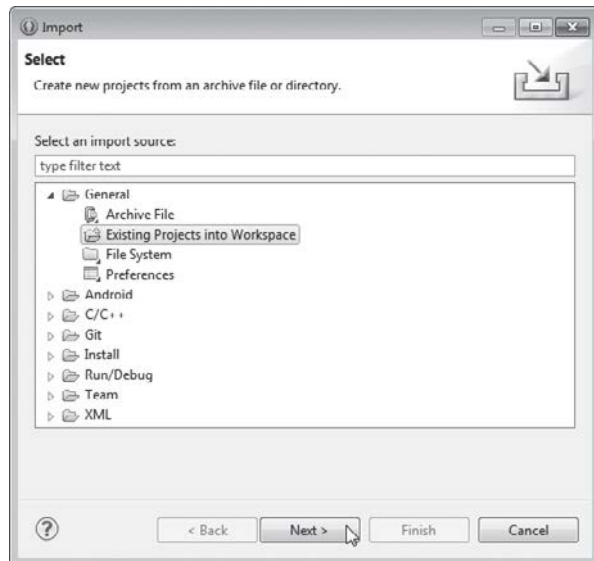


Figura 1.17 Importando um projeto já existente. (Parte I de 2.)

b) Passo **Import Projects**
da caixa de diálogo **Import**

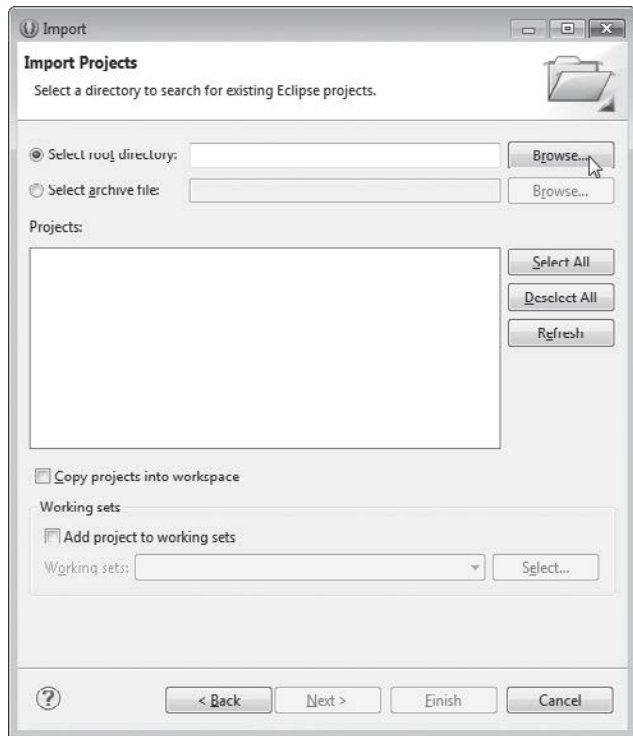


Figura 1.17 Importando um projeto já existente. (Parte 2 de 2.)

jects (Fig. 1.17(b)). Clique no botão **Browse...** à direita da caixa de texto **Select root directory**. Na caixa de diálogo **Browse For Folder**, localize a pasta **Doodlz** na pasta de exemplos do livro, selecione-a e clique em **Open**. Clique em **Finish** a fim de importar o projeto para o Eclipse. Agora o projeto aparece na janela **Package Explorer** (Fig. 1.18) no lado esquerdo do Eclipse. Se a janela **Package Explorer** não estiver visível, é possível vê-la selecionando **Window > Show View > Package Explorer**.

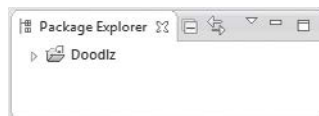


Figura 1.18 Janela **Package Explorer**.

6. **Ativando o aplicativo Doodlz.** No Eclipse, clique com o botão direito do mouse no projeto **Doodlz** na janela **Package Explorer** e, em seguida, selecione **Run As > Android Application** (Fig. 1.19). Isso executará o aplicativo **Doodlz** no AVD que você ativou no passo 4 (Fig. 1.20).
7. **Explorando o AVD e o modo imersivo.** Na parte inferior da tela do AVD, existem vários **botões programáveis** que aparecem na tela de toque do dispositivo. Você toca neles (usando o mouse em um AVD) para interagir com aplicativos e com o sistema operacional Android. O *botão voltar* retorna para a tela anterior do aplica-

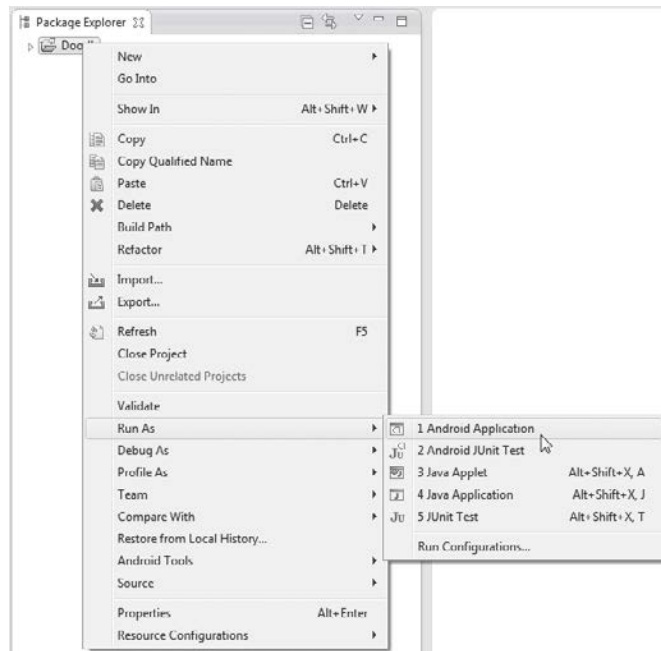


Figura 1.19 Ativando o aplicativo Doodlz.



Figura 1.20 Aplicativo Doodlz executando no AVD (Android Virtual Device).

tivo ou para um aplicativo anterior, caso você esteja na tela inicial do atual. O *botão Home* o leva de volta à tela inicial do dispositivo. O *botão de aplicativos recentes* permite ver a lista de aplicativos usados recentemente para que você possa voltar a eles rapidamente. Na parte superior da tela está a *barra de aplicativo*, a qual exibe o ícone e o nome do aplicativo e pode conter outros botões programáveis específicos – alguns aparecem na barra de aplicativo (**COLOR** e **LINE WIDTH** na Fig. 1.20) e o restante aparece no *menu de opções* do aplicativo (**I**). O número de opções na barra de aplicativo depende do tamanho do dispositivo – discutiremos isso no Capítulo 7. O Android 4.4 suporta um novo *modo imersivo* que permite aos aplicativos utilizar a tela inteira. Neste aplicativo, você pode tocar uma vez na área de desenho branca para ocultar as barras de status e de navegação do dispositivo, assim como a barra de ação do aplicativo. Você pode exibi-las novamente tocando outra vez na área de desenho ou fazendo um pressionamento forte (swipe) a partir da margem superior da tela.

8. **Entendendo as opções do aplicativo.** Para exibir as opções que não aparecem na barra de aplicativo, toque (isto é, clique) no ícone do menu de opções (**I**). A Figura 1.21(a) mostra a barra de ação e o menu de opções no AVD do Nexus 4, e a Fig. 1.21(b) os mostra no AVD de um Nexus 7 – as opções mostradas na barra de ação aparecem em letras maiúsculas pequenas. Tocar em **COLOR** exibe uma interface para mudar a cor da linha. Tocar em **LINE WIDTH** exibe uma interface para mudar a espessura da linha que vai ser desenhada. Tocar em **Eraser** configura a cor de desenho como branca para que, quando você desenhar sobre áreas coloridas, a cor seja apagada. Tocar em **Clear** primeiro confirma se você deseja apagar a imagem inteira e depois limpa a área de desenho caso a ação não seja cancelada. Tocar em **Save Image** salva a imagem na galeria (**Gallery**) de imagens do dispositivo. No Android 4.4, tocar em **Print** exibe uma interface para selecionar uma impressora disponível para imprimir sua imagem ou salvá-la como um documento PDF (o padrão). Você vai explorar cada uma dessas opções em breve.

9. **Mudando a cor do pincel para vermelha.** Para mudar a cor do pincel, primeiramente toque no item **COLOR** na barra de ação a fim de exibir a caixa de diálogo **Choose Color** (Fig. 1.22). As cores são definidas no *esquema de cores RGBA*, no qual

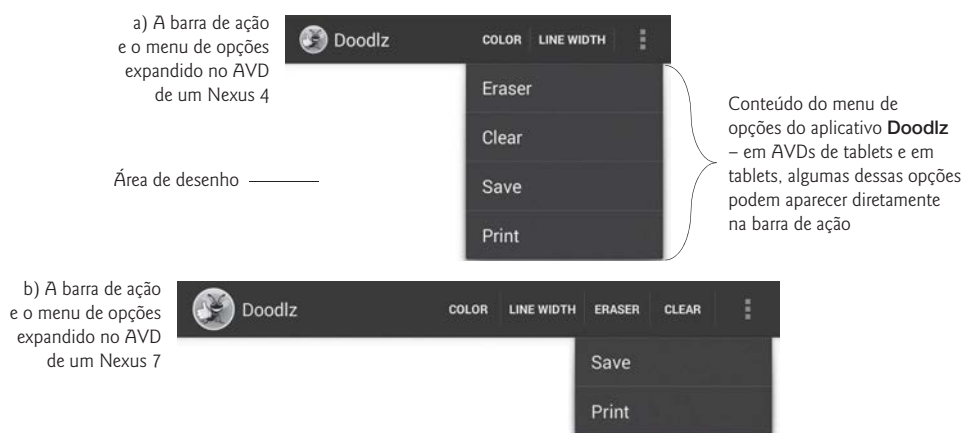


Figura 1.21 Menu de opções do aplicativo **Doodlz** expandido.

os componentes *alfa* (ou seja, *transparência*), vermelho, verde e azul são especificados por valores inteiros no intervalo 0 a 255. Para alfa, 0 significa *completamente transparente* e 255 significa *completamente opaco*. Para vermelho, verde e azul, 0 significa *nada* dessa cor e 255 significa a *quantidade máxima* dessa cor. A interface consiste nas barras de escolha (SeekBars) **Alpha**, **Red**, **Green** e **Blue**, que permitem selecionar a quantidade de alfa, vermelho, verde e azul na cor de desenho. Você arrasta as barras de escolha para mudar a cor. Quando você faz isso, o aplicativo exibe a nova cor abaixo das barras de escolha. Selecione uma cor vermelha agora, arrastando a barra de escolha **Red** para a direita, como na Figura 1.22. Toque no botão **Set Color** para voltar à área de desenho.

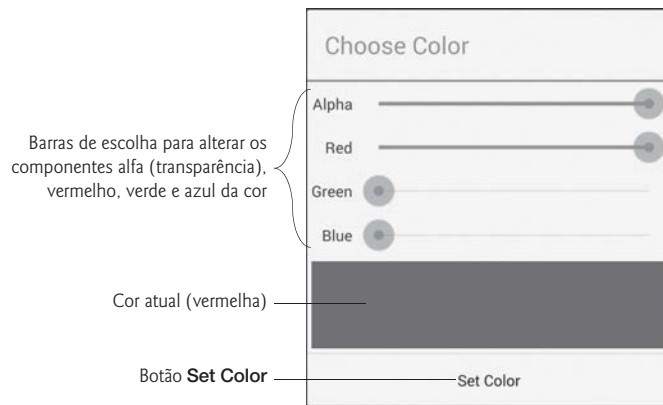


Figura 1.22 Mudando a cor de desenho para vermelha.

10. **Mudando a espessura da linha.** Para mudar a espessura da linha, toque em **LINE WIDTH** na barra de ação a fim de exibir a caixa de diálogo **Choose Line Width**. Arraste a barra de escolha da espessura de linha para a direita a fim de tornar a linha mais grossa (Fig. 1.23). Toque no botão **Set Line Width** para voltar à área de desenho.

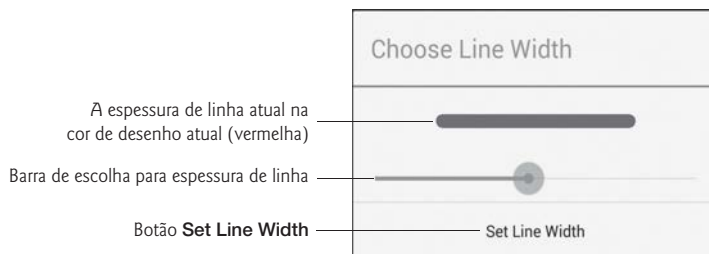


Figura 1.23 Mudando a espessura da linha.

11. **Desenhando as pétalas da flor.** Toque na tela para entrar no modo imersivo e, então, arraste seu “dedo” – o mouse ao usar o emulador – na área de desenho para desenhar pétalas de flor (Fig. 1.24).

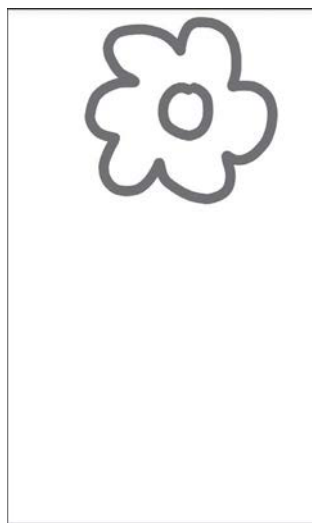


Figura 1.24 Desenhando pétalas de flor.

12. **Mudando a cor do pincel para verde-escuro.** Toque na tela para sair do modo imersivo e, então, toque em **COLOR** para exibir a caixa de diálogo **Choose Color**. Selecione um tom escuro de verde arrastando a barra de escolha **Green** para a direita e certificando-se de que as barras de escolha **Red** e **Blue** estejam na extremidade esquerda (Fig. 1.25(a)).

a) Selecionando verde-escuro como cor de desenho



b) Selecionando uma linha mais grossa

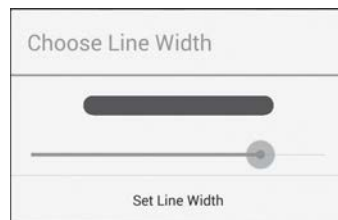


Figura 1.25 Mudando a cor para verde-escuro e tornando a linha mais grossa.

13. **Mudando a espessura da linha e desenhando o caule e as folhas.** Toque em **LINE WIDTH** para exibir a caixa de diálogo **Choose Line Width**. Arraste a barra de escolha da espessura da linha para a direita a fim de tornar a linha mais grossa (Fig.

1.25(b)). Toque na tela para entrar novamente no modo imersivo e, em seguida, desene o caule e as folhas da flor. Repita os passos 12 e 13 para uma cor verde mais clara e uma linha mais fina e, em seguida, desene a grama (Fig. 1.26).



Figura 1.26 Desenhando o caule e a grama.

14. **Finalizando o desenho.** Toque na tela para sair do modo imersivo. Em seguida, mude a cor do desenho para azul (Fig. 1.27(a)) e selecione uma linha mais estreita (Fig. 1.27(b)). Depois, toque na tela para entrar no modo imersivo e desene os pingos de chuva (Fig. 1.28).

a) Selecionando azul como cor de desenho



b) Selecionando uma linha mais fina

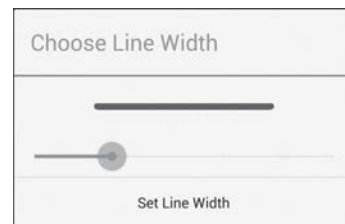


Figura 1.27 Mudando a cor para azul e estreitando a linha.



Figura 1.28 Desenhando a chuva na nova cor e espessura de linha.

15. **Salvando a imagem.** Você pode salvar sua imagem no aplicativo **Gallery** do dispositivo selecionando **Save** no menu de opções . Então, pode ver essa e outras imagens armazenadas no dispositivo abrindo o aplicativo **Gallery**.
16. **Imprimindo a imagem.** Para imprimir a imagem, selecione **Print** no menu de opções. Isso exibe a caixa de diálogo de impressão, a qual, por padrão, permite salvar a imagem como um documento PDF. Para selecionar uma impressora, toque em **Save as PDF** e selecione uma das impressoras disponíveis. Se não aparecer uma impressora na lista, talvez você precise configurar o Google Cloud Print para sua impressora. Para obter informações sobre isso, visite:

<http://www.google.com/cloudprint/learn/>

17. **Retornando à tela inicial.** Você pode voltar à tela inicial do AVD clicando no botão Home () no AVD. Para ver o desenho no aplicativo **Gallery**, toque em  para exibir a lista de aplicativos instalados no AVD. Então, você pode abrir o aplicativo **Gallery** para ver o desenho.

1.9.2 Executando o aplicativo Doodlz no AVD de um tablet

Para testar o aplicativo no AVD de um tablet, primeiramente ative o AVD, executando o passo 4 da Seção 1.9.1, mas selecione o AVD do Nexus 7 em vez do AVD do Nexus 4. Em seguida, clique com o botão direito do mouse no projeto **Doodlz** na janela **Package Explorer** do Eclipse e selecione **Run As > Android Application**. Se vários AVDs estiverem em execução quando você ativar um aplicativo, a caixa de diálogo **Android Device Chooser** (Fig. 1.29) aparecerá para permitir a escolha do AVD para instalar e executar o aplicativo. Nesse caso, os AVDs do Nexus 4 e do Nexus 7 estavam em execução em nosso sistema, de modo que havia dois dispositivos Android virtuais nos quais podíamos executar o aplicativo. Selecione o AVD do Nexus 7 e clique em **OK**. Esse aplicativo executa na orientação retrato (a largura é menor que a altura) no

telefone e em tablets pequenos. Se você executá-lo no AVD de um tablet grande (ou em um tablet grande), ele aparecerá na orientação paisagem (a largura é maior que a altura). A Figura 1.30 mostra o aplicativo executando no AVD do Nexus 7. Se o AVD for alto demais para exibir nossa tela, você pode mudar a orientação digitando *Ctrl* + *F12* (em um Mac, use *fn* + *control* + *F12*). Em alguns teclados, a tecla *Ctrl* se chama *Control*.

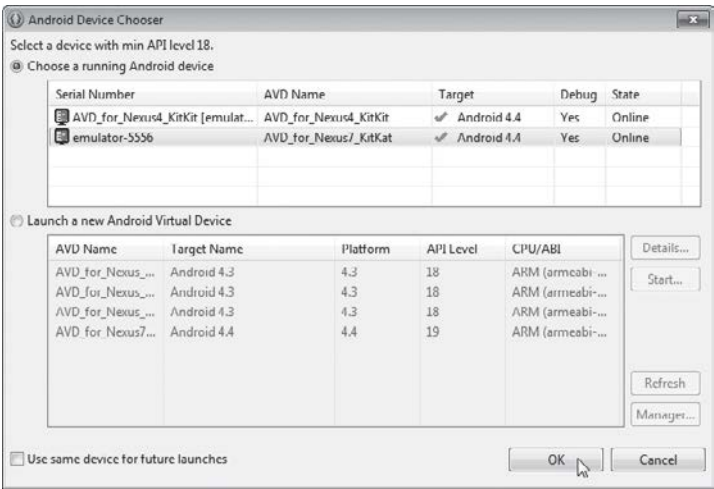


Figura 1.29 Caixa de diálogo Android Device Chooser.



Figura 1.30 Desenhando no AVD do Nexus 7.

1.9.3 Executando o aplicativo Doodlez em um aparelho Android

Caso você possua um aparelho Android, pode executar um aplicativo nele facilmente para propósitos de teste.

1. **Habilitando as opções de desenvolvedor no aparelho.** Primeiramente, você deve habilitar a depuração no aparelho. Para isso, acesse o aplicativo **Settings** do dispositivo e, então, selecione **About phone** (ou **About tablet**), localize o **Build number** (na parte inferior da lista) e toque nele repetidamente até ver a mensagem **You are now a developer** na tela. Isso habilitará uma entrada chamada **Developer options** no aplicativo **Settings**.
2. **Habilitando a depuração no aparelho.** Volte ao aplicativo **Settings**, selecione **Developer options** e certifique-se de que **USB debugging** esteja marcado – esse é o padrão ao habilitar as opções de desenvolvedor no dispositivo pela primeira vez.
3. **Conectando no dispositivo.** Em seguida, conecte o dispositivo em seu computador por meio do cabo USB que acompanha seu aparelho. Se você for usuário de Windows, lembre-se de que, conforme visto na seção “Antes de começar”, você talvez precise instalar um driver USB para seu dispositivo. Consulte as duas páginas web a seguir para saber os detalhes:

developer.android.com/tools/device.html
developer.android.com/tools/extras/oem-usb.html

4. **Executando o aplicativo Doodlz no aparelho Android.** No Eclipse, clique com o botão direito do mouse no projeto **Doodlz** na janela **Package Explorer** e, em seguida, selecione **Run As > Android Application**. Se você não tiver um AVD aberto, mas tiver um dispositivo Android conectado, o IDE vai instalar o aplicativo em seu dispositivo e executá-lo automaticamente. Caso você tenha um ou mais AVDs abertos e/ou dispositivos conectados, a caixa de diálogo **Android Device Chooser** (Fig. 1.29) vai aparecer para que o dispositivo ou AVD no qual o aplicativo que vai ser instalado e executado possa ser selecionado.

Preparando-se para distribuir aplicativos

Ao construir aplicativos para distribuição por meio das lojas de aplicativos, como o Google Play, você deve testá-los no máximo de dispositivos reais que puder. Lembre-se de que alguns recursos podem ser testados *somente* em dispositivos reais. Caso você não tenha muitos aparelhos disponíveis, crie AVDs que simulem os vários dispositivos nos quais gostaria de executar seu aplicativo. Ao configurar cada AVD para simular um dispositivo em particular, pesquise as especificações do aparelho online e configure o AVD de modo correspondente. Além disso, você pode modificar o arquivo `config.ini` do AVD, conforme descrito na Seção “**Setting hardware emulation options**”, no endereço

[developer.android.com/tools/devices/
managing-avds-cmdline.html#hardwareopts](http://developer.android.com/tools/devices/managing-avds-cmdline.html#hardwareopts)

Esse arquivo contém opções que não podem ser configuradas por meio do **Android Virtual Device Manager**. Modificando essas opções é possível fazê-las corresponder mais precisamente à configuração de hardware de um dispositivo real.

1.10 Construção de excelentes aplicativos Android

Com mais de 800 mil aplicativos no Google Play,⁹ como você faz para criar um aplicativo Android que as pessoas vão encontrar, baixar, usar e recomendar a outros? Reflita sobre o que torna um aplicativo divertido, útil, interessante, atraente e duradouro. Um nome de aplicativo engenhoso, um ícone interessante e uma descrição cativante podem

⁹ <http://www.pureoxymobile.com/how-many-apps-in-each-app-store/>.

atrair as pessoas para seu aplicativo no Google Play ou em uma das muitas outras lojas de aplicativos Android. Porém, uma vez que os usuários tenham baixado o aplicativo, o que os fará utilizá-lo regularmente e recomendá-lo a outros? A Figura 1.31 mostra algumas características de excelentes aplicativos.

Características de excelentes aplicativos

Jogos excelentes

- Interessantes e divertidos.
- Desafiadores.
- Níveis progressivos de dificuldade.
- Mostram sua pontuação e utilizam painéis de classificação para registrar as pontuações mais altas.
- Fornecem retorno sonoro e visual.
- Oferecem versões para um jogador, para vários jogadores e em rede.
- Possuem animações de alta qualidade.
- Descarregam código de entrada/saída e que utilizam muito poder de processamento para separar threads de execução a fim de melhorar os tempos de resposta da interface e o desempenho do aplicativo.
- Inovam com tecnologia de realidade aumentada, aprimorando um ambiente do mundo real com componentes virtuais; isso é particularmente popular em aplicativos baseados em vídeo.

Utilitários interessantes

- Fornecem funcionalidade útil e informações precisas.
- Aumentam a produtividade pessoal e empresarial.
- Tornam as tarefas mais convenientes (por exemplo, mantendo uma lista de tarefas, gerenciando despesas).
- Tornam o usuário mais bem informado.
- Fornecem informações atuais (por exemplo, a cotação mais recente de ações, notícias, alertas de tempestades fortes, atualizações do tráfego).
- Utilizam serviços baseados na localidade para fornecer serviços locais (por exemplo, cupons de empresas locais, melhores preços de combustíveis, entrega de alimentos).

Características gerais

- Em dia com os recursos mais recentes do Android, mas compatíveis com várias versões de Android para dar suporte ao maior público possível.
- Funcionam corretamente.
- Erros são corrigidos prontamente.
- Seguem as convenções padrão para interface gráfica de usuário de aplicativos Android.
- São ativados rapidamente.
- São rápidos nas respostas.
- Não exigem memória, largura de banda ou carga da bateria demais.
- São originais e criativos.
- Duradouros – algo que seus usuários utilizem regularmente.
- Usam ícones de qualidade profissional que aparecem no Google Play e no dispositivo do usuário.
- Usam elementos gráficos, imagens, animações, áudio e vídeo de qualidade.
- São intuitivos e fáceis de usar (não exigem documentação de ajuda extensa).
- Acessíveis a pessoas deficientes (<http://developer.android.com/guide/topics/ui/accessibility/index.html>).
- Dão aos usuários motivos e um significado para contar a outros sobre seu aplicativo (por exemplo, você pode dar aos usuários a opção de postar suas pontuações no Facebook ou no Twitter).

Figura 1.31 Características dos excelentes aplicativos. (continua)

Características de excelentes aplicativos

- Fornecem conteúdo adicional para aplicativos baseados em conteúdo (por exemplo, níveis de jogo, artigos, quebra-cabeças).
- Adaptados ao idioma (Capítulo 2) de cada país em que o aplicativo é oferecido (por exemplo, traduzir os arquivos de texto e de áudio do aplicativo, usar diferentes elementos gráficos de acordo com a localidade, etc.).
- Oferecem melhor desempenho, recursos e facilidade de uso do que os aplicativos concorrentes.
- Tiram proveito dos recursos internos do dispositivo.
- Não solicitam permissões em excesso.
- São projetados para serem executados da melhor maneira possível em uma ampla variedade de dispositivos Android.
- Preparados para o futuro em relação a novos dispositivos de hardware – especifique os recursos de hardware exatos utilizados por seu aplicativo para que o Google Play possa filtrá-lo e exibi-lo na loja apenas para dispositivos compatíveis (<http://android-developers.blogspot.com/2010/06/future-proofing-your-app.html>).

Figura 1.31 Características dos excelentes aplicativos.

1.1.1 Recursos para desenvolvimento com Android

A Figura 1.32 lista parte da importante documentação do site Android Developer (em inglês). À medida que se aprofundar no desenvolvimento de aplicativos Android, talvez você tenha perguntas sobre as ferramentas, questões de projeto, segurança e muito mais. Existem vários grupos de discussão e fóruns para desenvolvedores Android, em que você pode obter as notícias mais recentes ou fazer perguntas (Fig. 1.33). A Figura 1.34 lista vários sites em que você encontrará dicas, vídeos e recursos para desenvolvimento com Android.

Título	URL
<i>App Components</i>	http://developer.android.com/guide/components/index.html
<i>Using the Android Emulator</i>	http://developer.android.com/tools/devices/emulator.html
<i>Package Index</i>	http://developer.android.com/reference/packages.html
<i>Class Index</i>	http://developer.android.com/reference/classes.html
<i>Android Design</i>	http://developer.android.com/design/index.html
<i>Data Backup</i>	http://developer.android.com/guide/topics/data/backup.html
<i>Security Tips</i>	http://developer.android.com/training/articles/security-tips.html
<i>Managing Projects from Eclipse with ADT</i>	http://developer.android.com/guide/developing/projects/projects-eclipse.html
<i>Getting Started with Android Studio</i>	http://developer.android.com/sdk/installing/studio.html
<i>Debugging</i>	http://developer.android.com/tools/debugging/index.html
<i>Tools Help</i>	http://developer.android.com/tools/help/index.html
<i>Performance Tips</i>	http://developer.android.com/training/articles/perf-tips.html
<i>Keeping Your App Responsive</i>	http://developer.android.com/training/articles/perf-anr.html
<i>Launch Checklist (for Google Play)</i>	http://developer.android.com/distribute/googleplay/publish/preparing.html

Figura 1.32 Documentação online importante para desenvolvedores de Android. (continua)

Título	URL
<i>Get Started with Publishing</i>	http://developer.android.com/distribute/googleplay/publish/register.html
<i>Managing Your App's Memory</i>	http://developer.android.com/training/articles/memory.html
<i>Google Play Developer Distribution Agreement</i>	http://play.google.com/about/developer-distribution-agreement.html

Figura 1.32 Documentação online importante para desenvolvedores de Android.

Título	Assinatura	Descrição
Android Discuss	<i>Assine usando Google Groups:</i> android-discuss <i>Assine via e-mail:</i> android-discuss-subscribe@googlegroups.com	Um grupo de discussão geral sobre o Android, em que você pode obter respostas às suas perguntas sobre desenvolvimento de aplicativos.
Stack Overflow	http://stackoverflow.com/questions/tagged/android	Use essa lista para fazer perguntas sobre desenvolvimento de aplicativos Android em nível de iniciante, incluindo como começar a usar Java e Eclipse, e perguntas sobre as melhores práticas.
Android Developers	http://groups.google.com/forum/?fromgroups#!forum/android-developers	Os desenvolvedores Android experientes usam essa lista para solucionar problemas de aplicativos, problemas de projeto de interface gráfica do usuário, problemas de desempenho e muito mais.
Android Forums	http://www.androidforums.com	Faça perguntas, compartilhe dicas com outros desenvolvedores e encontre fóruns destinados a dispositivos Android específicos.

Figura 1.33 Grupos de discussão e fóruns sobre Android.

Dicas, vídeos e recursos para desenvolvimento com Android	URL
Exemplos de aplicativos Android do Google	http://code.google.com/p/apps-for-android/
Artigo da O'Reilly, "Ten Tips for Android Application Development"	http://answers.oreilly.com/topic/862-ten-tips-for-android-application-development/
Site Bright Hub™ para dicas sobre programação com Android e guias práticos	http://www.brighthub.com/mobile/google-android.aspx
O blog Android Developers	http://android-developers.blogspot.com/
O programa Sprint® Application Developers	http://developer.sprint.com/site/global/develop/mobile_platforms/android/android.jsp
Developer Center for Android do HTC	http://www.htcdev.com/
O site de desenvolvimento com Android da Motorola	http://developer.motorola.com/
Principais usuários de Android no Stack Overflow	http://stackoverflow.com/tags/android/topusers
Boletim semanal AndroidDev	http://androiddevweekly.com/
Blog Codependent de Chet Haase	http://graphics-geek.blogspot.com/
Blog sobre Android de Cyril Mottier	http://cyrilmottier.com/

Figura 1.34 Dicas, vídeos e recursos para desenvolvimento com Android. (continua)

Dicas, vídeos e recursos para desenvolvimento com Android	URL
Blog sobre Android de Romain Guy	http://www.curious-creature.org/category/android/
Canal no YouTube® para desenvolvedores de Android	http://www.youtube.com/user/androiddevelopers
Listas de reprodução de vídeo sobre Android	http://developer.android.com/develop/index.html
O que há de novo em ferramentas para desenvolvedor de Android	http://www.youtube.com/watch?v=1mv1dTnhLH4
Vídeos da sessão Google I/O 2013 Developer Conference	http://developers.google.com/events/io/sessions

Figura 1.34 Dicas, vídeos e recursos para desenvolvimento com Android.

1.12 Para finalizar

Este capítulo apresentou uma breve história do Android e examinou sua funcionalidade. Fornecemos links para parte da documentação online importante e para grupos de discussão e fóruns que você pode usar para se conectar à comunidade de desenvolvedores. Discutimos recursos do sistema operacional Android e fornecemos links para alguns aplicativos populares gratuitos e vendidos no Google Play. Apresentamos os pacotes Java, Android e Google que permitem usar as funcionalidades de hardware e software necessárias para construir uma variedade de aplicativos Android. Você vai usar muitos desses pacotes neste livro. Discutimos ainda a programação com Java e o SDK do Android. Você aprendeu sobre os gestos no Android e como fazer cada um deles em um dispositivo Android e no emulador. Oferecemos uma rápida revisão dos conceitos básicos da tecnologia de objetos, incluindo classes, objetos, atributos e comportamentos. Você testou o aplicativo **Doodlz** no emulador do Android para AVDs de smartphone e de tablet. No próximo capítulo, você vai construir seu primeiro aplicativo Android usando somente técnicas de programação visual. O aplicativo exibirá texto e duas imagens. Você também vai aprender sobre acessibilidade e internacionalização do Android.

Exercícios de revisão

1.1 Preencha os espaços em branco em cada um dos seguintes enunciados:

- Em 2007, a _____ foi formada para desenvolver, manter e aprimorar o Android, trazendo inovação para a tecnologia móvel, melhorando a experiência do usuário e reduzindo os custos.
- O Plugin _____ para Eclipse permite criar, executar e depurar aplicativos Android, e exportá-los para distribuição.
- As telas multitouch permitem controlar seu aparelho Android com _____ que envolvem apenas um toque ou vários toques simultâneos.
- Com web services, você pode criar _____, os quais permitem desenvolver aplicativos rapidamente, combinando web services complementares, frequentemente de diferentes organizações e possivelmente com outras formas de feeds de informação.
- O Android usa um conjunto de _____, que são grupos nomeados de classes predefinidas e relacionadas.
- O _____, incluído no SDK do Android, permite executar aplicativos Android em um ambiente simulado dentro do Windows, Mac OS X ou Linux.

- g) Praticamente qualquer substantivo pode ser representado como um objeto de software, em termos de _____ (por exemplo, nome, cor e tamanho) e comportamentos (por exemplo, cálculo, movimento e comunicação).
 - h) Uma unidade de programa, chamada _____, contém os métodos que executam as tarefas da classe.
 - i) Você envia mensagens para um objeto. Cada mensagem é uma _____ que diz a um método do objeto para que execute sua tarefa.
- 1.2** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- a) Uma vantagem de desenvolver aplicativos Android é que o sistema operacional é patentado pelo Google.
 - b) O grau de abertura da plataforma Android desestimula a inovação.
 - c) Você pode reutilizar uma classe muitas vezes para construir muitos objetos. A reutilização de classes já existentes ao construir novas classes e programas economiza tempo e trabalho.
 - d) Os atributos são especificados pelos métodos da classe.
 - e) Os objetos podem se comunicar entre si, mas normalmente não podem saber como outros objetos são implementados – os detalhes da implementação ficam ocultos dentro dos próprios objetos.
- 1.3** Preencha os espaços em branco em cada um dos seguintes enunciados (com base na seção 1.8):
- a) Os objetos têm a característica de _____ – embora alguns se comuniquem entre si, normalmente não podem saber como outros objetos são implementados.
 - b) As _____ de onde os objetos vêm são basicamente componentes de software reutilizáveis; elas incluem atributos e comportamentos.
 - c) O processo de analisar e projetar um sistema do ponto de vista orientado a objetos é denominado _____.
 - d) Com a _____, novas classes de objetos são derivadas, absorvendo características das classes existentes e adicionando suas próprias características exclusivas.
 - e) O tamanho, a forma, a cor e a espessura de um objeto são considerados _____ da classe do objeto.
 - f) Uma classe que representa uma conta bancária poderia conter um _____ para depositar dinheiro em uma conta, outro para sacar dinheiro de uma conta e um terceiro para informar o saldo da conta.
 - g) Você precisa construir um objeto de uma classe antes que um programa possa executar as tarefas definidas pelos métodos da classe — esse processo é chamado _____.
 - h) O saldo de uma classe conta bancária é um exemplo de _____ dessa classe.
 - i) Os requisitos de seu projeto definem o que o sistema deve fazer, e seu projeto especifica _____ o sistema deve fazer isso.

Respostas dos exercícios de revisão

- 1.1** a) Open Handset Alliance. b) ADT (Android Development Tools). c) gestos. d) mashups. e) pacotes. f) emulador do Android. g) atributos. h) classe. i) chamada de método.
- 1.2** a) Falsa. O sistema operacional é de código-fonte aberto e gratuito. b) Falsa. O grau de abertura da plataforma estimula a rápida inovação. c) Verdadeira. d) Falsa. Os atributos são especificados pelas variáveis de instância da classe. e) Verdadeira.
- 1.3** a) ocultação de informações. b) classes. c) análise e projeto orientados a objetos (OOAD – Object-Oriented Analysis and Design). d) herança. e) atributos. f) método. g) instanciamento. h) atributo. i) como.

Exercícios

- 1.4** Preencha os espaços em branco em cada um dos seguintes enunciados:
- a) Os aplicativos Android são desenvolvidos com _____ – uma das linguagens de programação mais usadas do mundo e uma escolha lógica, pois é poderosa, gratuita e de código-fonte aberto.
 - b) A programação de interfaces gráficas do usuário com Java é baseada em _____ – você vai escrever aplicativos que respondem a várias interações do usuário, como toques na tela e pressionamentos de tecla.
 - c) Tocar na tela e manter o dedo na posição é chamado de _____.
 - d) Tocar na tela e, depois, mover o dedo em uma direção e retirá-lo é chamado de _____.
 - e) Antes de executar um aplicativo no emulador, você precisa criar um _____, o qual define as características do dispositivo em que deseja fazer o teste, incluindo o hardware, a imagem do sistema, o tamanho da tela, o armazenamento de dados e muito mais.
 - f) Executar uma tarefa em um programa exige um _____ que contenha as instruções do programa que realmente executam suas tarefas.
 - g) Você precisa construir um objeto de uma classe antes que um programa possa executar as tarefas definidas pelos métodos da classe. Esse processo é chamado _____.
 - h) A _____ o ajuda a construir sistemas mais confiáveis e eficientes, pois as classes e componentes já existentes frequentemente passaram por extensivos testes, depuração e otimização de desempenho.
 - i) As classes _____ (isto é, empacotam) atributos e métodos nos objetos – os atributos e métodos de um objeto estão intimamente relacionados.
 - j) Uma nova classe de objetos pode ser criada rápida e convenientemente por meio de _____ – a nova classe absorve as características de outra já existente, possivelmente personalizando-as e adicionando suas próprias características exclusivas.
 - k) Ao contrário dos botões reais de um dispositivo, botões _____ aparecem na tela de toque do dispositivo.
 - l) As cores são definidas no esquema de cores ARGB, no qual os componentes vermelho, verde, azul e _____ são especificados por valores inteiros no intervalo 0 a 255.
- 1.5** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- a) A ampla maioria do desenvolvimento com Android é feita em C++.
 - b) O Microsoft Visual Studio é o ambiente de desenvolvimento integrado recomendado para desenvolvimento com Android, embora os desenvolvedores também possam usar um editor de texto e ferramentas de linha de comando para criar aplicativos Android.
 - c) No emulador, você pode reproduzir a maioria dos gestos e controles do Android usando o teclado e o mouse de seu computador.
 - d) Os objetos ou, mais precisamente, as classes de onde os objetos vêm, são basicamente componentes de software reutilizáveis.
- 1.6** Um dos objetos mais comuns é o relógio de pulso. Discuta como cada um dos termos e conceitos a seguir se aplica à noção de relógio: objeto, atributos, comportamentos, classe, herança (considere, por exemplo, um despertador), mensagens, encapsulamento e ocultação de informações.



Objetivos

Neste capítulo, você vai:

- Aprender os fundamentos do Android Developer Tools (o IDE do Eclipse e o Plugin ADT para escrever, testar e depurar seus aplicativos Android).
- Usar o IDE para criar um novo projeto de aplicativo.
- Projetar uma interface gráfica de usuário visualmente (sem programação) usando o editor **Graphical Layout** do IDE.
- Exibir texto e duas imagens em uma interface gráfica.
- Editar as propriedades de componentes da interface gráfica do usuário.
- Construir e ativar um aplicativo no emulador do Android.
- Tornar o aplicativo mais acessível para pessoas com deficiência visual, especificando strings para usar com os recursos TalkBack e Explore-by-Touch do Android.
- Dar suporte para internacionalização para que seu aplicativo possa exibir strings adaptadas para diferentes idiomas.

- 2.1 Introdução
- 2.2 Visão geral das tecnologias
 - 2.2.1 IDE Android Developer Tools
 - 2.2.2 Componentes `TextView` e `ImageView`
 - 2.2.3 Recursos do aplicativo
 - 2.2.4 Acessibilidade
 - 2.2.5 Internacionalização
- 2.3 Criação de um aplicativo
 - 2.3.1 Ativação do IDE Android Developer Tools
 - 2.3.2 Criação de um novo projeto
 - 2.3.3 Caixa de diálogo **New Android Application**
 - 2.3.4 Passo **Configure Project**
 - 2.3.5 Passo **Configure Launcher Icon**
 - 2.3.6 Passo **Create Activity**
 - 2.3.7 Passo **Blank Activity**
- 2.4 Janela Android Developer Tools
 - 2.4.1 Janela **Package Explorer**
 - 2.4.2 Janelas do editor
 - 2.4.3 Janela **Outline**
 - 2.4.4 Arquivos de recurso do aplicativo
 - 2.4.5 Editor **Graphical Layout**
 - 2.4.6 A interface gráfica de usuário padrão
- 2.5 Construção da interface gráfica de usuário do aplicativo com o editor **Graphical Layout**
 - 2.5.1 Adição de imagens ao projeto
 - 2.5.2 Alteração da propriedade **Id** dos componentes **RelativeLayout** e **TextView**
 - 2.5.3 Configuração do componente **TextView**
 - 2.5.4 Adição de componentes **ImageView** para exibir as imagens
- 2.6 Execução do aplicativo **Welcome**
- 2.7 Torne seu aplicativo acessível
- 2.8 Internacionalização de seu aplicativo
- 2.9 Para finalizar

Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios

2.1 Introdução

Neste capítulo, *sem escrever código*, você vai construir o aplicativo **Welcome**, o qual exibe uma mensagem de boas-vindas e duas imagens. Você vai usar o *IDE Android Developer Tools* para criar um aplicativo que pode ser executado em telefones Android. Em capítulos posteriores, veremos que também é possível criar aplicativos que podem

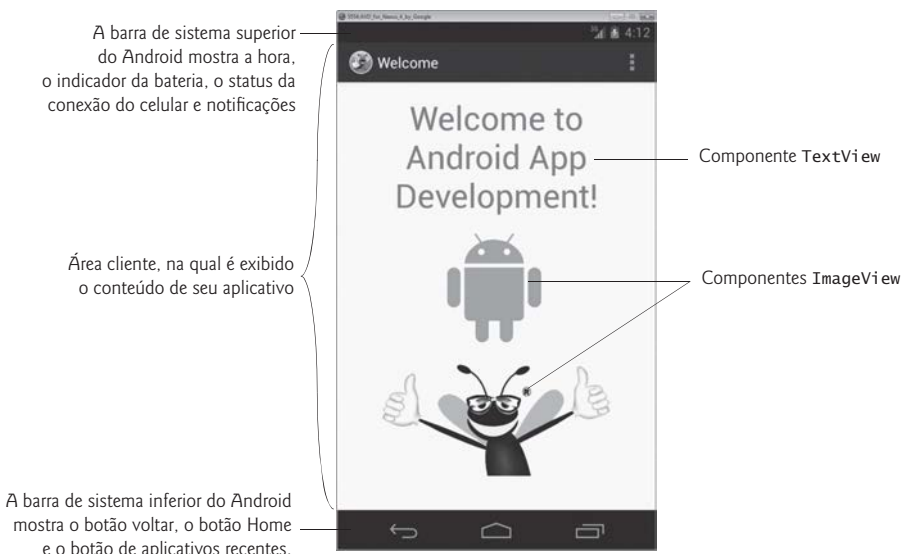


Figura 2.1 Aplicativo **Welcome** executando no emulador do Android.

ser executados em tablets ou em telefones e em tablets. Você vai criar um aplicativo Android simples (Fig. 2.1) usando o **editor Graphical Layout** do IDE, o qual permite construir interfaces gráficas utilizando técnicas de *arrastar e soltar*. Vai executar seu aplicativo no *emulador* do Android (e em um telefone Android, caso tenha um disponível). Por fim, vai aprender a tornar o aplicativo mais *acessível* para pessoas deficientes e a *internacionalizá-lo* para exibir strings *adaptadas* para diferentes idiomas. No site do livro – <http://www.deitel.com/books/AndroidHTP2> –, fornecemos a versão para *IDE Android Studio* deste capítulo (em inglês). Aqui, presumimos que você tenha lido o Prefácio, a seção “Antes de começar” e a Seção 1.9.

2.2 Visão geral das tecnologias

Esta seção apresenta as tecnologias que você vai aprender neste capítulo.

2.2.1 IDE Android Developer Tools

Este capítulo apresenta o *IDE Android Developer Tools*. Ele vai ser utilizado para criar um novo projeto (Seção 2.3). Conforme você vai ver, o IDE cria a interface gráfica do usuário padrão que contém o texto “Hello world!”. Em seguida, você vai usar o editor **Graphical Layout** do IDE e a janela **Properties** para construir visualmente uma interface gráfica de usuário simples, consistindo em texto e duas imagens (Seção 2.5).

2.2.2 Componentes TextView e ImageView

O texto deste aplicativo é exibido em um componente **TextView**, e suas imagens são exibidas em componentes **ImageView**. A interface gráfica padrão criada para este aplicativo contém um componente **TextView** que você vai modificar utilizando a janela **Properties** do IDE para configurar várias opções, como o texto do componente **TextView**, o tamanho da fonte e a cor da fonte (Seção 2.5.3). Em seguida, você usará o componente **Palette** dos controles da interface do usuário do editor **Graphical Layout** para arrastar e soltar componentes **ImageView** na interface (Seção 2.5.4).

2.2.3 Recursos do aplicativo

Considera-se uma boa prática definir todas as strings e valores numéricos em arquivos de recurso, os quais são colocados nas subpastas da pasta *res* de um projeto. Na Seção 2.5.3, você vai aprender a criar recursos para strings (como o texto de um componente **TextView**) e medidas (como o tamanho da fonte). Vai aprender também a usar um recurso de cor interno do Android para especificar a cor da fonte do componente **TextView**.

2.2.4 Acessibilidade

O Android contém muitos recursos de *acessibilidade* para ajudar pessoas com vários tipos de deficiências a usar seus dispositivos. Por exemplo, deficientes visuais e físicos podem usar o recurso **TalkBack** do Android para permitir que um dispositivo pronuncie o texto da tela ou um texto que você forneça para ajudá-los a entender o objetivo e o conteúdo de um componente da interface gráfica. O recurso **Explore by Touch** do Android permite ao usuário tocar na tela para ouvir o aplicativo **TalkBack** falar o que está na tela próximo ao local do toque. A Seção 2.7 mostra como habilitar esses recursos e como configurar os componentes da interface gráfica do usuário de seu aplicativo para acessibilidade.

2.2.5 Internacionalização

Os dispositivos Android são usados no mundo todo. Para atingir o maior público possível com seus aplicativos, você deve pensar em personalizá-los para várias *localidades* e idiomas falados – isso é conhecido como **internacionalização**. A Seção 2.8 mostra como fornecer texto em espanhol para as strings de acessibilidade dos componentes `TextView` e `ImageView` do aplicativo **Welcome**; em seguida, mostra como testar o aplicativo em um AVD configurado para o idioma espanhol.

2.3 Criação de um aplicativo

Os exemplos deste livro foram desenvolvidos com as versões do Android Developer Tools (versão 22.x) e do SDK do Android (versões 4.3 e 4.4) que eram as mais atuais quando a obra estava sendo produzida. Supomos que você tenha lido a seção “Antes de começar” e tenha configurado o Java SE Development Kit (JDK) e o IDE Android Developer Tools que utilizou no teste da Seção 1.9. Esta seção mostra como usar o IDE para criar um novo projeto. Vamos apresentar recursos adicionais do IDE por todo o livro.

2.3.1 Ativação do IDE Android Developer Tools

Para ativar o IDE, abra a subpasta eclipse da pasta de instalação do *Android SDK/ADT bundle* e, em seguida, clique duas vezes no ícone do Eclipse (☼ ou ☿, dependendo de sua plataforma). Quando o IDE é iniciado pela primeira vez, aparece a página **Welcome** (mostrada originalmente na Fig. 1.14). Se ela não aparecer, selecione **Help > Android IDE** para exibi-la.

2.3.2 Criação de um novo projeto

Um **projeto** é um grupo de arquivos relacionados, como arquivos de código e imagens que compõem um aplicativo. Para criar um aplicativo, você deve primeiro criar seu projeto. Para isso, clique no botão **New Android Application...** na página **Welcome** a fim de exibir a **caixa de diálogo New Android Application** (Fig. 2.2). Você também pode fazer isso selecionando **File > New > Android Application Project** ou clicando na lista suspensa do botão **New** (📁) da barra de ferramentas e selecionando **Android Application Project**.

2.3.3 Caixa de diálogo New Android Application

No primeiro passo da caixa de diálogo **New Android Application** (Fig. 2.2), especifique as seguintes informações e depois clique em **Next >**:

1. Campo **Application Name**: – o nome de seu aplicativo. Digite **Welcome** nesse campo.
2. Campo **Project Name**: – o nome do projeto, o qual aparece no nó-raiz do projeto, na guia **Package Explorer** do IDE. Por padrão, o IDE configura isso com o nome do aplicativo *sem espaços* e com cada letra inicial maiúscula – para um aplicativo chamado **Address Book**, o nome do projeto seria **AddressBook**. Se preferir usar outro nome, digite-o no campo **Project Name**.
3. Campo **Package Name**: – o nome do pacote Java para o código-fonte de seu aplicativo. O Android e a loja Google Play utilizam isso como *identificador exclusivo* do aplicativo, o qual deve permanecer o mesmo em *todas* as versões de seu aplicativo. O nome do pacote normalmente começa com o nome de domínio de sua empresa ou instituição *ao contrário* – o nosso é **deitel.com**, de modo que iniciamos nossos nomes de pacote com **com.deitel**. Em geral, isso é seguido pelo nome do aplicativo. Por convenção,

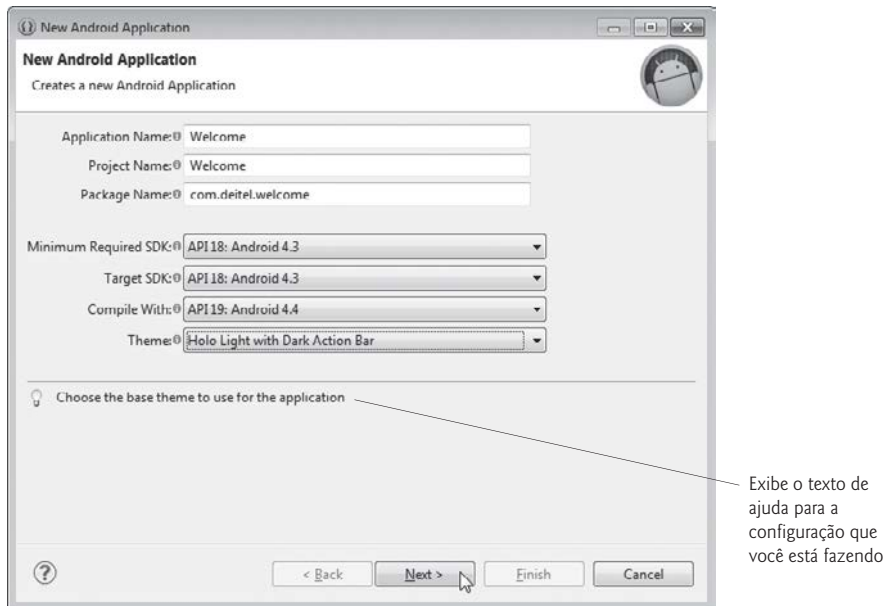


Figura 2.2 Caixa de diálogo New Android Application.

os nomes de pacote utilizam apenas letras minúsculas. O IDE especifica um nome de pacote que começa com `com.example` por padrão – isso serve *apenas* para propósitos de aprendizagem e deve ser alterado caso você pretenda distribuir seu aplicativo.

4. Campo **Minimum Required SDK**: – o *nível de API do Android mínimo* exigido para executar seu aplicativo. Isso permite que ele seja executado em dispositivos nesse nível de API e *mais altos*. Usamos o nível de API 18, que corresponde ao Android 4.3 – o mais baixo das duas versões utilizadas neste livro. A Figura 2.3 mostra as versões do SDK do Android e os níveis de API. Outras versões do SDK atualmente estão obsoletas (*deprecated*) e *não* devem ser usadas. A porcentagem de dispositivos Android executando cada versão da plataforma aparece em:

<http://developer.android.com/about/dashboards/index.html>

Versão do SDK	Nível de API	Versão do SDK	Nível de API	Versão do SDK	Nível de API
4.4	19	4.0.3–4.0.4	15	2.2	8
4.3	18	4.0.1	14	2.1	7
4.2.x	17	3.2	13	1.6	4
4.1.x	16	2.3.3–2.3.7	10		

Figura 2.3 Versões do SDK do Android e níveis de API (<http://developer.android.com/about/dashboards/index.html>).

5. Campo **Target SDK**: – o nível de API *preferido* para seu aplicativo. Usamos o nível 19 para os aplicativos deste livro. Quando esta obra estava sendo produzida, 26% dos dispositivos Android ainda usavam o nível 10. Ao desenvolver aplicativos para distribuição, você frequentemente quer atingir o máximo de dispositivos

possível. Por exemplo, para atingir dispositivos com Android 2.3.3 e superiores (98% de todos os dispositivos Android), você configuraria **Minimum Required SDK** como 10. Se for configurado com um nível de API anterior ao que está em **Target SDK**, *you should certify that your application does not use resources of API levels above Minimum Required SDK or that it can detect the API level of the device and adjust its functionality accordingly*. A ferramenta *Android Lint*, que o IDE executa em segundo plano, indica os recursos não suportados que estão sendo utilizados.

6. Campo **Compile With:** – a versão da API usada ao compilar seu aplicativo. Normalmente, é igual ao **Target SDK**, mas poderia ser uma versão anterior que suporte todas as APIs utilizadas em seu aplicativo.
7. Campo **Theme:** – o tema padrão do Android de seu aplicativo, o qual proporciona ao aplicativo aparência e comportamento compatíveis com o Android. Existem três temas a escolher – *Holo Light*, *Holo Dark* e *Holo Light with Dark Action Bars* (o padrão especificado pelo IDE). Ao projetar uma interface gráfica do usuário, você pode escolher dentre muitas variações dos temas *Holo Light* e *Holo Dark*. Para este capítulo, usaremos o tema padrão – vamos discutir os temas com mais detalhes em capítulos subsequentes. Para obter mais informações sobre cada tema e ver exemplos de capturas de tela, visite

<http://developer.android.com/design/style/themes.html>

2.3.4 Passo Configure Project

No passo **Configure Project** da caixa de diálogo **New Android Application** (Fig. 2.4), deixe as configurações padrão como mostrado e clique em **Next >**. Essas configurações permitem especificar o ícone para seu aplicativo e definir a **Activity** – uma classe que controla a execução do aplicativo – nos passos subsequentes.

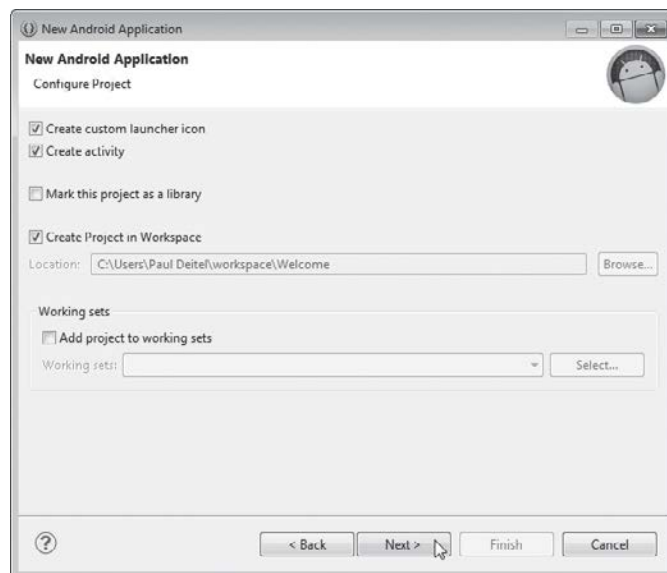


Figura 2.4 Caixa de diálogo **New Android Application** – passo 2 **New Android Application**.

2.3.5 Passo Configure Launcher Icon

Quando seu aplicativo é instalado em um dispositivo, seu ícone e seu nome aparecem, com todos os outros aplicativos instalados, no *lançador*, o qual você pode acessar por meio do ícone  na tela inicial de seu dispositivo. O Android pode ser executado em uma grande variedade de dispositivos, com diferentes tamanhos e resoluções de tela. Para garantir uma boa aparência de suas imagens em todos os dispositivos, você deve fornecer várias versões de cada imagem utilizada por seu aplicativo. O Android pode escolher a imagem correta automaticamente, com base em diversas especificações, como a resolução da tela (largura e altura em pixels) ou DPI (pontos por polegada). Vamos discutir esses mecanismos a partir do Capítulo 3. Mais informações sobre projetos para tamanhos e resoluções de tela variados podem ser encontradas em

<http://developer.android.com/training/multiscreen/index.html>

e sobre ícones de modo geral em

<http://developer.android.com/design/style/iconography.html>

O passo **Configure Launcher Icon** (Fig. 2.5) permite configurar o ícone do aplicativo a partir de uma imagem existente, de um clip-art ou de um texto. Isso exige que você especifique e crie versões em escala de 48 por 48, 72 por 72, 96 por 96 e 144 por 144 para suportar várias resoluções de tela. Para este aplicativo, usamos uma imagem chamada *Deite10range.png*. Para usá-la, clique em **Browse...** à direita do campo **Image File**; navegue até a pasta *images* na pasta de exemplos do livro, selecione *Deite10range.png* e clique em **Open**. Apresentações prévias das imagens em escala aparecem na área **Preview** da caixa de diálogo. Essas imagens serão colocadas nas pastas apropriadas no projeto do aplicativo. Nem sempre as imagens são bem dimensionadas. Para aplicativos que você pretende colocar na loja Google Play, talvez queira pedir para um artista projetar ícones para as resoluções adequadas. No Capítulo 9, discutimos o envio de aplicativos para a loja Google Play e listamos várias empresas que oferecem serviços de projeto de ícones gratuitos e pagos. Clique em **Next >** para continuar no passo **Create Activity**.

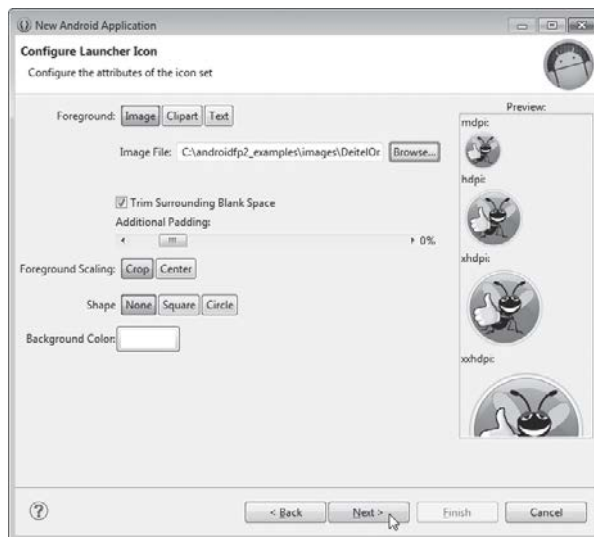


Figura 2.5 Caixa de diálogo **New Android Application** – passo **Configure Launcher Icon**.

2.3.6 Passo Create Activity

No passo **Create Activity** (Fig. 2.6), você seleciona o template para a Activity de seu aplicativo. Os **templates** economizam seu tempo, fornecendo pontos de partida previamente configurados para projetos de aplicativo comumente utilizados. A Figura 2.7 descreve sucintamente três dos templates mostrados na Figura 2.6. Para este aplicativo, selecione **Blank Activity** e, em seguida, clique em **Next >**. Vamos usar os outros templates em capítulos posteriores.

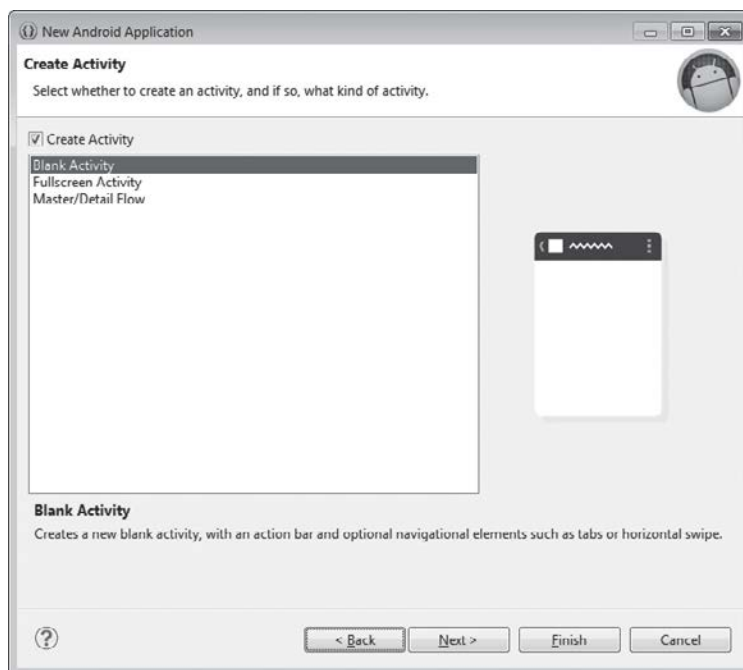


Figura 2.6 Caixa de diálogo **New Android Application** – passo **Create Activity**.

Template	Descrição
Blank Activity	Utilizado para um <i>aplicativo de tela única</i> , no qual você mesmo constrói a maior parte da interface gráfica do usuário. Fornece uma <i>barra de ação</i> na parte superior do aplicativo, a qual exibe o nome do aplicativo, e pode mostrar controles que permitem ao usuário interagir com ele.
Fullscreen Activity	Utilizado para um <i>aplicativo de tela única</i> (semelhante a Blank Activity) que ocupa a tela inteira, mas pode alternar a visibilidade da barra de status do dispositivo e a barra de ação do aplicativo.
Master/Detail Flow	Utilizado para um aplicativo que exibe uma <i>lista mestra</i> de itens, dos quais o usuário pode escolher um para ver seus <i>detalhes</i> – semelhante aos aplicativos incorporados Email e People . Para tablets, a lista mestra e os detalhes aparecem lado a lado na mesma tela. Para telefones, ela é mostrada em uma única tela, e a seleção de um item exibe seus detalhes em uma tela separada.

Figura 2.7 Templates de Activity.

2.3.7 Passo Blank Activity

Este passo depende do template selecionado no passo anterior. Para o template **Blank Activity**, este passo permite especificar:

- **Activity Name** (nome da atividade) – `MainActivity` é o nome padrão fornecido pelo IDE. Esse é o nome de uma subclasse de `Activity` que controla a execução do aplicativo. A partir do Capítulo 3, vamos modificar essa classe para implementar as funcionalidades de um aplicativo.
- **Layout Name** (nome do layout) – `activity_main.xml` é o nome de arquivo padrão fornecido pelo IDE. Esse arquivo armazena uma representação em XML da interface gráfica do usuário do aplicativo. Neste capítulo, você vai construir a interface do usuário (Seção 2.5) usando técnicas visuais.
- **Fragment Layout Name** (nome do layout do fragmento) – `fragment_main` é o nome de arquivo padrão fornecido pelo IDE. A interface gráfica do usuário (GUI) de uma atividade normalmente contém um ou mais fragmentos que descrevem porções dessa GUI. No template de aplicativo padrão, `activity_main` mostra a GUI descrita por `fragment_main`. Discutimos fragmentos em mais detalhes a partir do Capítulo 5. Até lá, vamos simplesmente ignorar o arquivo `fragment_main`.
- **Navigation Type** (tipo de navegação) – `None` é o padrão especificado pelo IDE. O aplicativo **Welcome** não fornece nenhuma funcionalidade. Em um aplicativo que suporta interações do usuário, você pode selecionar um **Navigation Type** apropriado para permitir que o usuário navegue pelo conteúdo de seu aplicativo. Vamos discutir as opções de navegação com mais detalhes em aplicativos posteriores.

Clique em **Finish** para criar o projeto.



Figura 2.8 Caixa de diálogo **New Android Application** – passo **Blank Activity**.

2.4 Janela Android Developer Tools

Depois de criar o projeto, o IDE abre `MainActivity.java` e `fragmente_main.xml`. Feche esse último arquivo e abra `activity_main.xml` a partir da pasta `res/layout` do projeto para que o IDE apareça como mostrado na Figura 2.9. O IDE mostra o editor **Graphical Layout** para que você possa começar a projetar a interface gráfica do usuário de seu aplicativo. Neste capítulo, discutimos apenas os recursos do IDE necessários para construir o aplicativo **Welcome**. Vamos apresentar muito mais recursos do IDE ao longo do livro.



Figura 2.9 O projeto **Welcome** aberto na janela Android Developer Tools.

2.4.1 Janela Package Explorer

A janela **Package Explorer** dá acesso a todos os arquivos do projeto. A Figura 2.10 mostra o projeto do aplicativo **Welcome** na janela **Package Explorer**. O nó **Welcome** representa o

projeto. É possível ter muitos projetos abertos simultaneamente no IDE – cada um vai ter seu próprio *nó de nível superior*. Dentro de um nó do projeto, o conteúdo é organizado em pastas e arquivos. Neste capítulo, você vai usar apenas arquivos localizados na pasta *res*, a qual discutimos na Seção 2.4.4 – discutiremos as outras pastas à medida que as utilizarmos em capítulos posteriores.

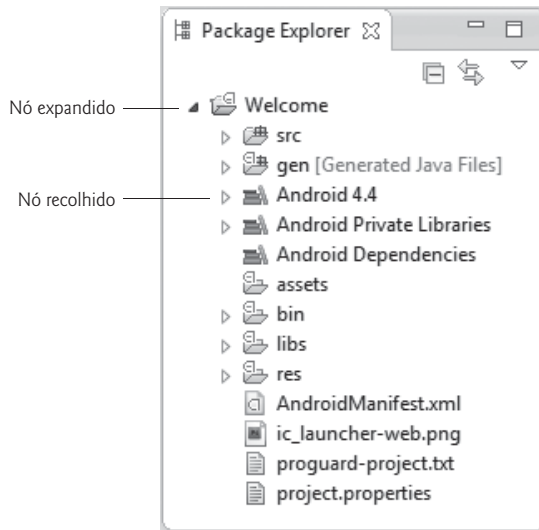


Figura 2.10 Janela Package Explorer.

2.4.2 Janelas do editor

À direita do **Package Explorer** na Figura 2.9 está a janela do editor **Graphical Layout**. Quando você clica duas vezes em um arquivo no **Package Explorer**, seu conteúdo aparece em uma janela de editor apropriada, dependendo do tipo do arquivo. Para um arquivo Java, aparece o editor de código-fonte Java. Para um arquivo XML que representa uma interface gráfica do usuário (como `activity_main.xml`), aparece o editor **Graphical Layout**.

2.4.3 Janela Outline

A janela **Outline** é exibida no lado direito do IDE (Fig. 2.9). Essa janela mostra informações relacionadas ao arquivo que está sendo editado no momento. Para uma interface gráfica de usuário, ela mostra todos os elementos que compõem a interface. Para uma classe Java, ela mostra o nome da classe e seus métodos e campos.

2.4.4 Arquivos de recurso do aplicativo

Arquivos de layout, como `activity_main.xml` (na pasta `res/layout` do projeto), são considerados *recursos* de aplicativo e são armazenados na pasta *res* do projeto. Dentro dessa pasta, existem subpastas para diferentes tipos de recurso. Os que utilizamos neste aplicativo aparecem na Figura 2.11, e os outros (`menu`, `animator`, `anim`, `color`, `raw` e `xml`) serão discutidos à medida que precisarmos deles ao longo do livro.

Subpasta de recurso	Descrição
<code>drawable</code>	Nomes de pasta que começam com <code>drawable</code> normalmente contêm imagens. Essas pastas também podem conter arquivos XML representando formas e outros tipos de itens desenháveis (como as imagens que representam os estados <i>não pressionado</i> e <i>pressionado</i> de um botão).
<code>layout</code>	Nomes de pasta que começam com <code>layout</code> contêm arquivos XML que descrevem interfaces gráficas do usuário, como o arquivo <code>activity_main.xml</code> .
<code>values</code>	Nomes de pasta que começam com <code>values</code> contêm arquivos XML que especificam valores de <i>arrays</i> (<code>arrays.xml</code>), <i>cores</i> (<code>colors.xml</code>), <i>dimensões</i> (<code>dimen.xml</code> ; valores como larguras, alturas e tamanhos de fonte), <i>strings</i> (<code>strings.xml</code>) e <i>estilos</i> (<code>styles.xml</code>). Esses nomes de arquivo são usados por convenção, mas <i>não</i> são obrigatórios – na verdade, você pode colocar todos os recursos desses tipos em <i>um</i> arquivo apenas. Considera-se a melhor prática definir os dados de arrays, cores, tamanhos, strings e estilos codificados no programa como <i>recursos</i> , para que possam ser facilmente modificados sem alterar o código Java do aplicativo. Por exemplo, se um <i>recurso de dimensão</i> é referenciado em muitos locais em seu código, você pode alterar o arquivo de recurso apenas uma vez, em vez de localizar todas as ocorrências de um valor de dimensão codificado nos arquivos-fonte Java de seu aplicativo.

Figura 2.11 Subpastas da pasta `res` do projeto utilizadas neste capítulo.

2.4.5 Editor Graphical Layout

Quando você cria um projeto, o IDE abre o arquivo `Fragment_main.xml` do aplicativo no editor **Graphical Layout**. Se ainda não fez isso, feche esse arquivo e clique duas vezes em `activity_main.xml` na pasta `res/layout` de seu aplicativo para abri-lo no editor (Fig. 2.12).

Selecionando o tipo de tela para o projeto da interface gráfica do usuário

Os dispositivos Android podem ser executados em muitos tipos de aparelhos. Neste capítulo, você vai projetar a interface gráfica do usuário de um telefone Android. Conforme mencionamos na seção “Antes de começar”, utilizamos para esse propósito um AVD que emula o telefone Google Nexus 4. O editor **Graphical Layout** vem com muitas configurações de dispositivo que representam vários tamanhos e resoluções de tela que podem ser usados para projetar sua interface. Para este capítulo, usamos o **Nexus 4** predefinido, o qual selecionamos na lista suspensa de tipo de tela na Figura 2.12. Isso não quer dizer que o aplicativo só pode ser executado em um aparelho Nexus 4 – significa simplesmente que o projeto é para dispositivos com tamanho e resolução de tela similares aos do Nexus 4. Em capítulos posteriores, você vai aprender a projetar suas interfaces gráficas para mudar de escala adequadamente para uma ampla variedade de dispositivos.

2.4.6 A interface gráfica de usuário padrão

A interface gráfica de usuário padrão (Fig. 2.12) para um aplicativo **Blank Page** consiste em um componente `FrameLayout` (chamado **container**) com fundo claro (especificado pelo tema que escolhemos ao criarmos o projeto). Um **FrameLayout** é projetado para mostrar apenas um componente de interface gráfica do usuário – normalmente um layout que contém muitos outros componentes de interface. Neste aplicativo, usaremos **RelativeLayout**, um componente que organiza os elementos da interface gráfica de usuário *um em relação ao outro* ou *em relação ao layout em si* – por exemplo, você pode especificar que um elemento deve aparecer *abaixo* de outro e ser *centralizado horizon-*

A **Palette** contém **Widgets** (componentes de interface gráfica de usuário), **Layouts** e outros itens que podem ser arrastados e soltos na tela de desenho

A lista suspensa de tipo de tela especifica dispositivos para os quais você quer fazer seu projeto de interface gráfica de usuário – para este capítulo, selecione **Nexus 4**

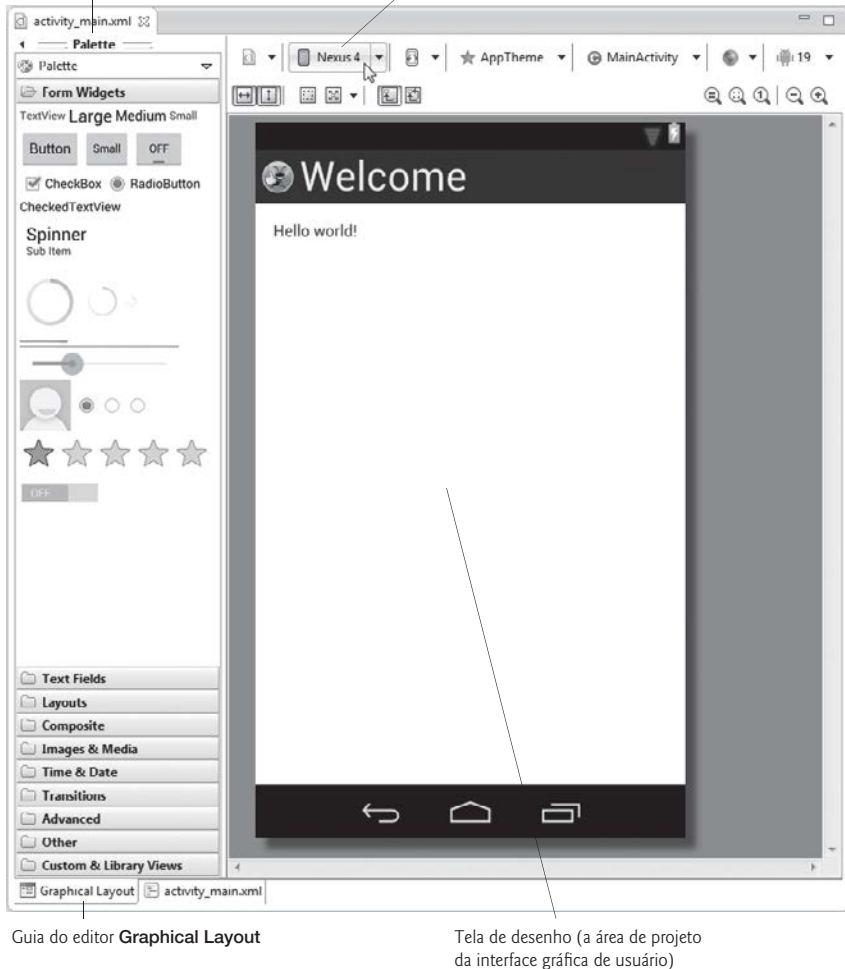


Figura 2.12 Vista do editor **Graphical Layout** da interface gráfica de usuário padrão do aplicativo.

talmente dentro de `RelativeLayout`. Um componente `TextView` exibe texto. Vamos falar mais sobre cada um deles na Seção 2.5.

2.5 Construção da interface gráfica de usuário do aplicativo com o editor **Graphical Layout**

O editor **Graphical Layout** do IDE permite construir a interface gráfica do usuário arrastando e soltando componentes – como `TextView`, `ImageView` e `Button` – em uma área de projeto. Por padrão, o layout da interface gráfica de um `MainActivity` de **Blank App** é armazenado em um arquivo XML chamado `activity_main.xml`, localizado na pasta `res`

do projeto, na subpasta layout. Neste capítulo, vamos usar o editor **Graphical Layout** e a janela **Outline** para construir a interface gráfica e *não* vamos estudar o código XML gerado. As ferramentas de desenvolvimento para Android foram aprimoradas a ponto de, na maioria dos casos, não ser necessário manipular a marcação XML diretamente.

2.5.1 **Adição de imagens ao projeto**

Para este aplicativo, você precisará adicionar ao projeto a imagem do inseto da Deitel (bug.png) e a imagem do logotipo do Android (android.png). Elas se encontram com os exemplos do livro na subpasta Welcome da pasta images. *Todos os nomes de arquivo de recursos de imagem – e todos os outros recursos sobre os quais você vai aprender em capítulos posteriores – devem ter letras minúsculas.*

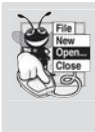
Como os dispositivos Android têm *tamanhos de tela, resoluções e densidades de pixel* (isto é, pontos por polegada ou DPI – dots per inch) variados, você normalmente fornece imagens em resoluções variadas, que o sistema operacional escolhe com base na densidade de pixels de um dispositivo. Por isso, a pasta res de seu projeto contém várias subpastas que começam com o nome drawable. Essas pastas armazenam imagens com diferentes densidades de pixels (Fig. 2.13).

Densidade	Descrição
drawable-ldpi	Densidade baixa – aproximadamente 120 pontos por polegada.
drawable-mdpi	Densidade média – aproximadamente 160 pontos por polegada.
drawable-hdpi	Densidade alta – aproximadamente 240 pontos por polegada.
drawable-xhdpi	Densidade extra alta – aproximadamente 320 pontos por polegada.
drawable-xxhdpi	Densidade extra extra alta – aproximadamente 480 pontos por polegada.
drawable-xxxhdpi	Densidade extra extra extra alta – aproximadamente 640 pontos por polegada.

Figura 2.13 Densidades de pixels do Android.

As imagens para dispositivos com densidade de pixels semelhante ao telefone Google Nexus 4 que utilizamos em nosso AVD de telefone são colocadas na pasta drawable-hdpi. As imagens para dispositivos com densidades de pixels mais altas (como as de alguns telefones e tablets) são colocadas nas pastas drawable-xhdpi ou drawable-xxhdpi. As imagens para as telas de densidade média e baixa de dispositivos Android mais antigos são colocadas nas pastas drawable-mdpi e drawable-ldpi, respectivamente.

Para este aplicativo, fornecemos somente uma versão de cada imagem. Se o Android não conseguir encontrar uma imagem na pasta drawable apropriada, aumentará ou diminuirá a escala da versão a partir de outra pasta drawable, de acordo com as diferentes densidades.



Observação sobre aparência e comportamento 2.1

Imagens em baixa resolução não proporcionam boa escalabilidade. Para que as imagens sejam bem visualizadas, um dispositivo de alta densidade de pixels precisa de imagens de resolução mais alta do que um dispositivo de baixa densidade de pixels.



Observação sobre aparência e comportamento 2.2

Para obter informações detalhadas sobre suporte para várias telas e tamanhos de tela no Android, visite http://developer.android.com/guide/practices/screens_support.html.

Execute os passos a seguir para adicionar as imagens nesse projeto:

1. Na janela **Package Explorer**, expanda a pasta **res** do projeto.
2. Localize e abra a subpasta **Welcome** da pasta **images** em seu sistema de arquivos e, então, arraste as imagens para a subpasta **drawable-hdpi** da pasta **res**. Na caixa de diálogo **File Operation** que aparece, certifique-se de que **Copy Files** esteja selecionado e, então, clique em **OK**. Em geral, você deve usar imagens PNG, mas também são aceitas imagens JPG e GIF.

Agora essas imagens podem ser usadas no aplicativo.

2.5.2 Alteração da propriedade **Id** dos componentes **RelativeLayout** e **TextView**

Quando uma interface gráfica de usuário é exibida no editor **Graphical Layout**, você pode usar a janela **Properties** na parte inferior da janela **Outline** para configurar as propriedades do layout ou do componente selecionado sem editar a marcação XML diretamente. Para selecionar um layout ou componente, clique nele no editor **Graphical Layout** ou selecione seu nó na janela **Outline** (Fig. 2.14). Em geral, é mais fácil selecionar componentes específicos na janela **Outline**.

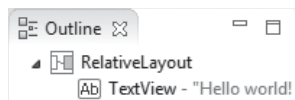


Figura 2.14 Visualização hierárquica da interface gráfica do usuário na janela **Outline**.

Para começar a construir a interface gráfica do usuário, clique com o botão direito em **FrameLayout**, na janela **Outline**, selecione **ChangeLayout...**, depois **RelativeLayout** e clique em **OK**.

Você deve mudar o nome de cada layout e componente para um nome relevante, especialmente se eles serão manipulados via programação (como faremos em aplicativos posteriores). Isso é feito por meio da **propriedade Id** – a **Id** padrão para o **FrameLayout** em **activity_main.xml** é **container**, o que iremos mudar. A propriedade **Id** pode ser usada para acessar e modificar um componente em um layout e a partir de um código Java. Conforme veremos em breve, **Id** é usada para especificar o *posicionamento relativo* de elementos em um componente **RelativeLayout**. Na parte superior da janela **Properties**, configure seu valor como

```
@+id/welcomeRelativeLayout
```

e pressione **Enter**. Na caixa de diálogo **Update References**, clique em **Yes**; na caixa de diálogo **Rename Resource**, clique em **OK** para completar a alteração. O **+** na sintaxe **@+id** indica que uma *nova id* para referenciar aquele componente de interface gráfica deve ser criada usando o identificador à direita do sinal de barra normal (**/**). A janela **Properties** deve agora aparecer como na Figura 2.15.

Na maioria dos aplicativos, é preciso dar espaço extra em torno de um layout – algo conhecido como *padding* – para separar os componentes do layout de componentes de outros layouts e das bordas da tela do dispositivo. Com as mudanças recentes no template de aplicativo padrão do Google, esse *padding* não é mais incluído em **activity_main.xml**. Para adicioná-lo a este aplicativo, role até a subseção **View** da janela **Properties**. Para as propriedades **Padding Left** e **Padding Right**, clique no botão de elipse, selecione

`activity_horizontal_margin` e clique em **OK**. Repita esses passos para **Padding Top** e **Padding Bottom**, selecionando `activity_vertical_margin`. O *padding* será discutido em mais detalhes no próximo capítulo.

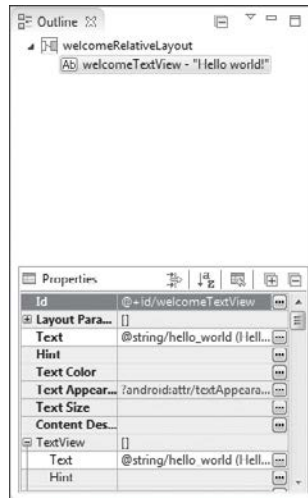


Figura 2.15 Janela **Properties** após a alteração das propriedades **Id** dos componentes **RelativeLayout** e **TextView** na interface gráfica de usuário padrão do aplicativo.

2.5.3 Adição e configuração do componente **TextView**

Adicionando o componente *TextView* e configurando sua propriedade *Id*

Para adicionar o componente **TextView** à interface gráfica, arraste um **TextView** da **Paleta** no lado esquerdo do editor **Graphical Layout** até o nó `welcomeRelativeLayout` na janela **Outline**. Por padrão, o IDE atribui a **TextView** a identificação `textView1`. Com o **TextView** selecionado na janela **Outline**, mude sua propriedade **Id** para

```
@+id/welcomeTextView
```

Configurando a propriedade *Text* de *TextView* usando um recurso de *string*

De acordo com a documentação do Android para recursos de aplicativo

```
http://developer.android.com/guide/topics/resources/index.html
```

considera-se uma boa prática colocar strings, arrays de string, imagens, cores, tamanhos de fonte, dimensões e outros recursos de aplicativo no arquivo XML dentro das subpastas da pasta `res` do projeto, para que os recursos possam ser gerenciados separadamente do código Java de seu aplicativo. Esse processo é conhecido como *externalizar* os recursos. Por exemplo, se você externaliza valores de cor, todos os componentes que utilizam a mesma cor podem ser atualizados com uma nova cor simplesmente alterando-se o valor da cor em um arquivo de recursos centralizado.

Se quiser *adaptar* seu aplicativo para vários idiomas, então armazenar as strings *separadamente* do código do aplicativo permitirá alterá-las facilmente. Na pasta `res` de seu projeto, a subpasta `values` contém um arquivo `strings.xml` que é utilizado para armazenar as strings do idioma padrão do aplicativo – inglês, para nossos aplicativos. Para fornecer strings adaptadas para outros idiomas, você pode criar pastas `values` separadas para cada idioma, conforme demonstraremos na Seção 2.8.

Para configurar a **propriedade Text** de `TextView`, crie um novo recurso de string no arquivo `strings.xml`, como segue:

1. Certifique-se de que o componente `welcomeTextView` esteja selecionado.
2. Localize sua propriedade **Text** na janela **Properties** e clique no botão de reticências à direita do valor da propriedade para exibir a caixa de diálogo **Resource Chooser**.
3. Na caixa de diálogo **Resource Chooser**, clique no botão **New String...** para exibir a caixa de diálogo **Create New Android String** (Fig. 2.16).

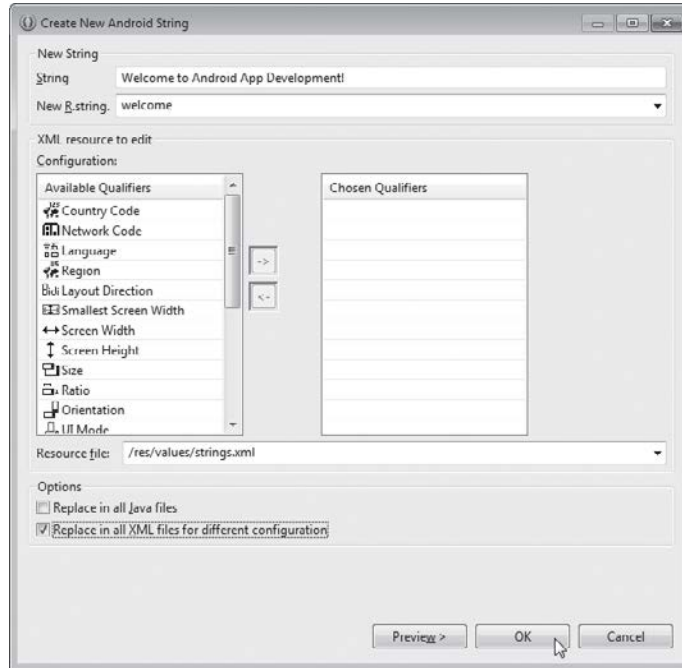


Figura 2.16 Caixa de diálogo **Create New Android String**.

4. Preencha os campos **String** e **New R.string** como mostrado na Fig. 2.16, marque a caixa de seleção **Replace in all XML file for different configurations** e, então, clique em **OK** para fechar a caixa de diálogo e retornar à caixa de diálogo **Resource Chooser**. O campo **String** especifica o texto que será exibido no componente `TextView`, e o campo **R.string** especifica o nome do recurso de string para que possamos referenciá-lo na propriedade **Text** de `TextView`.
5. O novo recurso de string, chamado `welcome`, é selecionado automaticamente. Clique em **OK** na caixa de diálogo **Resource Chooser** para utilizar esse recurso.

Na janela **Properties**, a propriedade **Text** deve agora aparecer como na Fig. 2.17. A sintaxe `@string` indica que um recurso de string será selecionado no arquivo `strings.xml` (localizado na pasta `res/values` do projeto), e `welcome` indica o recurso de string a ser selecionado.

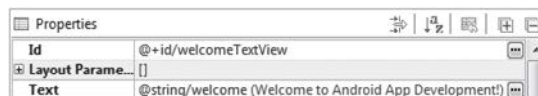


Figura 2.17 Janela **Properties** após a alteração da propriedade **Text** de `TextView`.

Configurando a propriedade *Text Size* de *TextView* – pixels em escala e pixels independentes da densidade

O tamanho dos componentes da interface gráfica do usuário e do texto pode ser especificado em várias unidades de medida (Fig. 2.18). A documentação para suportar vários tamanhos de tela

http://developer.android.com/guide/practices/screens_support.html

recomenda usar *pixels independentes de densidade* para as dimensões dos componentes da interface gráfica do usuário e outros elementos de tela, e *pixels independentes de escala* para tamanhos de fonte.

Unidade	Descrição
px	pixel
dp ou dip	pixel independente de densidade
sp	pixel independente de escala
in	polegadas
mm	milímetros

Figura 2.18 Unidades de medida.

Definir suas interfaces gráficas com **pixels independentes de densidade** permite à plataforma Android *mudar a escala* da interface com base na densidade de pixels da tela de determinado dispositivo. Um *pixel independente de densidade* é equivalente a um pixel em uma tela com 160 dpi. Em uma tela de 240 dpi, cada pixel independente de densidade vai mudar de escala por um fator de 240/160 (isto é, 1,5). Assim, um componente com 100 *pixels independentes de densidade* de largura vai mudar de escala para 150 *pixels* de largura *reais*. Em uma tela com 120 dpi, cada pixel independente de densidade muda de escala por um fator de 120/160 (isto é, 0,75). Assim, o mesmo componente com 100 pixels independentes de densidade de largura vai ter 75 pixels de largura reais. Os *pixels independentes de escala* mudam de escala como os pixels independentes de densidade, e também mudam de acordo com o *tamanho de fonte preferido* do usuário (conforme especificado nas configurações do dispositivo).

Agora você vai aumentar o tamanho da fonte de *TextView* e adicionar algum preenchimento acima desse componente para separar o texto da borda da tela do dispositivo. Para mudar o tamanho da fonte:

1. Certifique-se de que o componente `welcomeTextView` esteja selecionado.
2. Localize sua **propriedade Text Size** na janela **Properties** e clique no botão de reticências à direita do valor da propriedade para exibir a caixa de diálogo **Resource Chooser**.
3. Na caixa de diálogo **Resource Chooser**, clique no botão **New Dimension...**
4. Na caixa de diálogo que aparece, especifique `welcome_textsize` para **Name** e 40sp para **Value**; em seguida, clique em **OK** para fechar a caixa de diálogo e retornar à caixa de diálogo **Resource Chooser**. As letras sp no valor 40sp indicam que essa é uma medida de *pixel independente da escala*. As letras dp em um valor de dimensão (por exemplo, 10dp) indicam uma medida de *pixel independente de densidade*.
5. O novo recurso de dimensão chamado `welcome_textsize` é selecionado automaticamente. Clique em **OK** para usar esse recurso.

Configurando mais propriedades de TextView

Use a janela **Properties** para especificar as seguintes propriedades adicionais de TextView:

- Configure sua **propriedade Text Color** como `@android:color/holo_blue_dark`. O Android tem vários recursos de cor predefinidos. Quando você digita `@android:color/` no campo de valor da propriedade **Text Color**, aparece uma lista suspensa com recursos de cor (Fig. 2.19). Selecione `@android:color/holo_blue_dark` nessa lista para mudar o texto para azul-escuro.

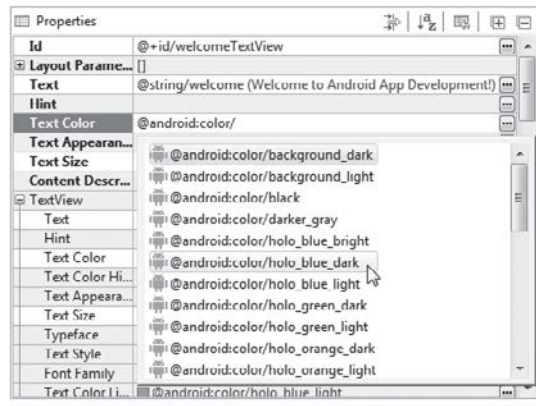


Figura 2.19 Configurando a propriedade **Text Color** de um componente TextView como `@android:color/holo_blue_dark`.

- Para centralizar o texto no componente TextView, caso ele utilize várias linhas, configure sua **propriedade Gravity** como `center`. Para isso, clique no campo **Value** dessa propriedade e, em seguida, clique no botão de reticências para exibir a caixa de diálogo **Select Flags Values** com as opções da propriedade **Gravity** (Fig. 2.20). Clique na caixa de seleção `center` e, em seguida, clique em **OK** para definir o valor.

Agora a janela do editor **Graphical Layout** deve aparecer como na Figura 2.21.

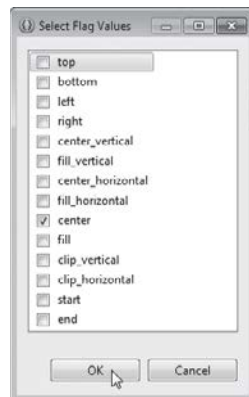


Figura 2.20 Opções para a propriedade **Gravity** de um objeto.



Figura 2.21 A interface gráfica do usuário após a conclusão da configuração do componente `TextView`.

2.5.4 Adição de componentes `ImageView` para exibir as imagens

A seguir, você vai adicionar dois componentes `ImageView` à interface gráfica do usuário para exibir as imagens que adicionou ao projeto na Seção 2.5.1. Vai fazer isso arrastando os componentes `ImageView` da seção **Images & Media** da *Palette* para a interface, abaixo do componente `TextView`. Para isso, execute os passos a seguir:

1. Expanda a categoria **Images & Media** da *Palette* e, então, arraste um componente `ImageView` para a área de desenho, como mostrado na Fig. 2.22. O novo componente `ImageView` aparece abaixo do nó `welcomeTextView`. Quando você arrasta um componente para a área de desenho, o editor **Graphical Layout** exibe *marcadores de régua verdes* e aparece uma dica de ferramenta (tooltip). Os marcadores ajudam a posicionar os componentes na interface do usuário. A dica de ferramenta mostra como o componente da interface será configurado se você soltá-lo na posição atual do mouse. A dica de ferramenta na Fig. 2.22 indica que o componente `ImageView` será *centralizado horizontalmente* no layout pai (também indicado pelo marcador de régua tracejado que se estende de cima para baixo na interface do usuário) e colocado abaixo do componente `welcomeTextView` (também indicado por uma seta pelo marcador de régua tracejado).
2. Quando o componente `ImageView` é solto, a caixa de diálogo **Resource Chooser** (Fig. 2.23) aparece para que você possa escolher o recurso de imagem a ser exibido. Para cada imagem colocada em uma pasta `drawable`, o IDE gera um ID de recurso (isto é, um nome de recurso) que pode ser usado para referenciar essa imagem em seu projeto de interface gráfica de usuário e no código. O ID de recurso é o nome de arquivo da imagem, sem a extensão – para `android.png`, o ID de recurso é `android`. Selecione `android` e clique em **OK** a fim de exibir a imagem do robô. Quando um novo componente é adicionado à interface do usuário, ele é selecionado automaticamente e suas propriedades aparecem na janela **Properties**.



Figura 2.22 Arrastando e soltando um componente `ImageView` na interface gráfica do usuário.



Figura 2.23 Selecionando o recurso de imagem `android` na caixa de diálogo **Resource Chooser**.

3. O IDE configura a propriedade `id` do novo componente `ImageView` como `@+id/imageView1` por padrão. Altere isso para `@+id/droidImageView`. A caixa de diálogo **Update References?** aparece para confirmar a operação de *mudança de nome*. Clique em **Yes**. Em seguida, aparece a caixa de diálogo **Rename Resource** para mostrar

todas as alterações que serão feitas. Clique em **OK** para concluir a operação de mudança de nome.

4. Repita os passos 1 a 3 anteriores a fim de criar o componente `bugImageView`. Para esse componente, arraste o elemento `ImageView` para baixo de `droidImageView`, selecione o recurso de imagem do inseto na caixa de diálogo **Resource Chooser** e configure a propriedade `Id` como `@+id/bugImageView` na janela **Properties**; em seguida, salve o arquivo.

Agora a interface gráfica do usuário deve aparecer como a mostrada na Figura 2.24.

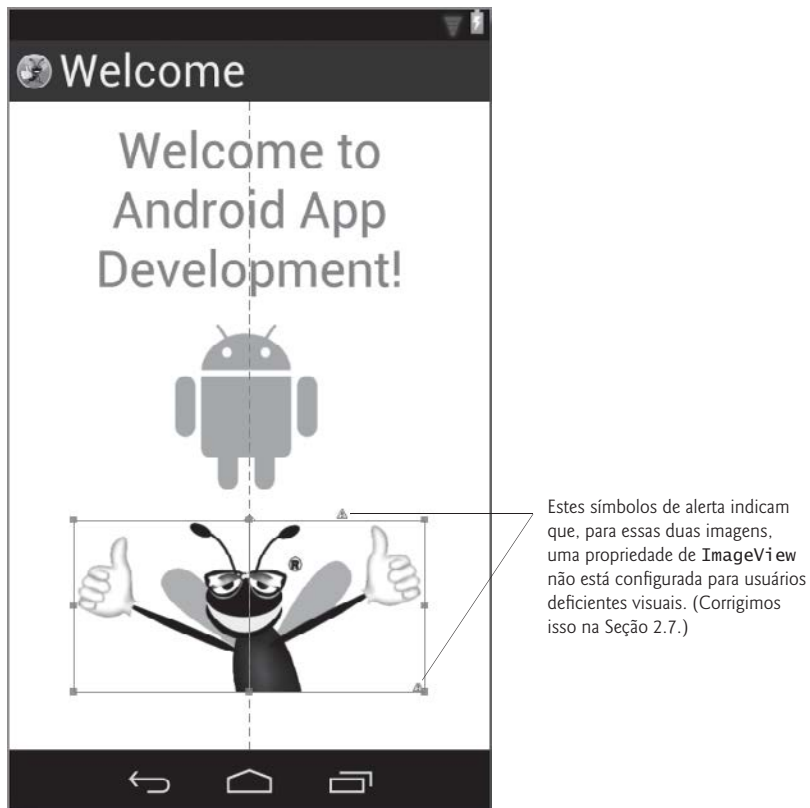


Figura 2.24 Projeto da interface gráfica do usuário concluído.

2.6 Execução do aplicativo Welcome

Para executar o aplicativo em um *AVD* (*Android Virtual Device*) para um telefone, execute os passos mostrados na Seção 1.9.1. A Figura 2.25 mostra o aplicativo em execução no AVD do Nexus 4 que foi configurado na seção “Antes de começar”. Ele aparece na orientação *retrato*, em que a altura do dispositivo é maior que sua largura. Embora seja possível girar seu dispositivo ou AVD para a orientação *paisagem* (em que a largura é maior que a altura), a interface gráfica desse aplicativo não foi projetada para essa orientação. No próximo capítulo, você vai aprender a restringir a orientação de um aplicativo

e, em capítulos subsequentes, vai aprender a criar interfaces de usuário mais dinâmicas, que podem lidar com as duas orientações.

Se quiser, você pode seguir os passos da Seção 1.9.3 para executar o aplicativo em um dispositivo Android. Embora esse aplicativo possa ser executado em um AVD de tablet ou em um tablet Android, sua interface gráfica ocupará somente uma pequena parte da tela de um tablet. Em geral, para aplicativos que podem ser executados tanto em telefones como em tablets, você também fornecerá um layout de tablet que utilize melhor o espaço disponível na tela, conforme demonstraremos em capítulos posteriores.

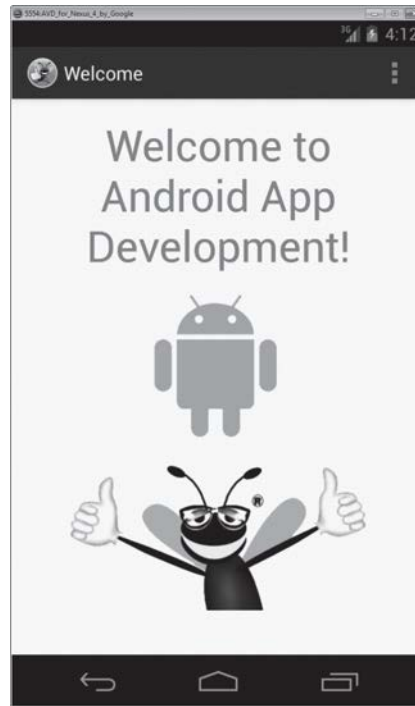


Figura 2.25 Aplicativo **Welcome** executado em um AVD.

2.7 Torne seu aplicativo acessível

O Android contém recursos de *acessibilidade* para ajudar pessoas com vários tipos de deficiências a usar seus dispositivos. Para deficientes visuais, o recurso **TalkBack** do Android pode pronunciar o texto que está na tela ou que você fornece (ao projetar sua interface de usuário ou por meio de programa) para ajudar o usuário a entender a finalidade de um componente da interface. O Android fornece também o recurso **Explore by Touch**, o qual permite ao usuário ouvir o aplicativo TalkBack pronunciar o que está no local em que o usuário toca na tela.

Quando o aplicativo TalkBack está habilitado e o usuário toca em um componente acessível da interface do usuário, ele pronuncia o texto de acessibilidade do componente e vibra o dispositivo para dar um retorno para deficientes auditivos. Todos os componentes de interface de usuário padrão do Android oferecem suporte para acessibilidade. Para os que exibem texto, o recurso TalkBack pronuncia esse texto por padrão – por

exemplo, quando o usuário toca em um componente `TextView`, o TalkBack pronuncia o texto desse componente. O recurso TalkBack é habilitado no aplicativo **Settings**, sob **Accessibility**. Nessa página também é possível habilitar outros recursos de acessibilidade do Android, como um *tamanho de texto padrão maior* e a capacidade de usar *gestos que ampliam áreas da tela*. Infelizmente, atualmente o recurso TalkBack *não* é suportado em AVDs, de modo que você precisa executar esse aplicativo em um dispositivo para ouvi-lo pronunciar o texto. Quando o recurso TalkBack é habilitado, o Android oferece a opção de percorrer um tutorial sobre como utilizá-lo com o recurso Explore by Touch.

Habilitando o recurso TalkBack dos componentes `ImageView`

No aplicativo **Welcome**, não precisamos de texto mais descritivo para o componente `TextView`, pois o recurso TalkBack lerá o conteúdo desse componente. Contudo, para um componente `ImageView`, não há texto para o recurso TalkBack pronunciar, a não ser que você o forneça. Considera-se uma melhor prática no Android garantir que *todo* componente de interface gráfica de usuário possa ser usado com o recurso TalkBack, fornecendo-se texto para a **propriedade Content Description** de todo componente que não exiba texto. Por isso, o IDE nos avisou de que algo deu errado em nossa interface de usuário, exibindo pequenos ícones de alerta () no editor **Graphical Layout**, ao lado de cada componente `ImageView`. Esses alertas – os quais são gerados por uma ferramenta do IDE conhecida como *Android Lint* – indicam que não configuramos a propriedade **Content Description** de cada imagem. O texto que você fornece deve ajudar o usuário a entender a finalidade do componente. Para um componente `ImageView`, o texto deve descrever a imagem.

Para adicionar uma propriedade **Content Description** para cada componente `ImageView` (e eliminar os alertas da *Android Lint*), execute os passos a seguir:

1. Selecione o componente `droidImageView` no editor **Graphical Layout**.
2. Na janela **Properties**, clique no botão de reticências à direita da propriedade **Content Description** a fim de abrir a caixa de diálogo **Resource Chooser**.
3. Clique no botão **New String...** para exibir a caixa de diálogo **Create New Android String**.
4. Especifique “Android logo” no campo **String** e, no campo **R.string**, especifique `android_logo`; em seguida, pressione **OK**.
5. O novo recurso de string `android_logo` é selecionado na caixa de diálogo **Resource Chooser**; portanto, clique em **OK** para especificar esse recurso como valor para a propriedade **Content Description** de `droidImageView`.
6. Repita os passos anteriores para o componente `bugImageView`, mas na caixa de diálogo **Create New Android String**, especifique “Deite1 double-thumbs-up bug logo” para o campo **String** e “`deite1_logo`” para o campo **R.string**. Salve o arquivo.

Quando você define a propriedade **Content Description** de cada componente `ImageView`, o ícone de alerta () do componente é removido no editor **Graphical Layout**.

Testando o aplicativo com TalkBack habilitado

Execute esse aplicativo em um dispositivo com TalkBack habilitado e, então, toque no componente `TextView` e em cada componente `ImageView` para ouvir o recurso TalkBack pronunciar o texto correspondente.

Aprendendo mais sobre acessibilidade

Alguns aplicativos geram componentes de interface gráfica dinamicamente, em resposta às interações do usuário. Para esses componentes, você pode definir o texto de acessibilidade via programação. As páginas de documentação para desenvolvedores de Android listadas a seguir fornecem mais informações sobre os recursos de acessibilidade do Android e uma lista de pontos a seguir ao desenvolver aplicativos acessíveis:

```
http://developer.android.com/design/patterns/accessibility.html
http://developer.android.com/guide/topics/ui/accessibility/index.html
http://developer.android.com/guide/topics/ui/accessibility/checklist.html
```

2.8 Internacionalização de seu aplicativo

Como você sabe, os dispositivos Android são usados no mundo todo. Para atingir o maior público possível, você deve pensar em personalizar seus aplicativos para várias localidades e idiomas falados – isso é conhecido como **internacionalização**. Por exemplo, se você pretende oferecer seu aplicativo na França, deve traduzir seus recursos (por exemplo, arquivos de texto e áudio) para o francês. Você também poderia optar por usar diferentes cores, elementos gráficos e sons, com base na *localidade*. Para cada localidade, você vai ter um conjunto separado e personalizado de recursos. Quando o usuário ativa o aplicativo, o Android localiza e carrega automaticamente os recursos correspondentes às configurações de localidade do dispositivo.

Adaptação local (localização)

Uma vantagem importante de definir seus valores de string como recursos de string (como fizemos neste aplicativo) é que você pode *adaptar* (ou “localizar”) facilmente seu aplicativo para a localidade, criando arquivos de recurso XML adicionais para esses recursos de string em outros idiomas. Em cada arquivo, você usa os mesmos nomes de recurso de string, mas fornece a string *traduzida*. O Android pode então escolher o arquivo de recursos apropriado, com base no idioma preferido do usuário do dispositivo.

Dando nomes às pastas de recursos adaptados para o local

Os arquivos de recurso XML que contêm strings adaptadas ao local são colocados em subpastas da pasta `res` do projeto. O Android usa um esquema de atribuição de nomes de pasta especial para escolher automaticamente os recursos adaptados ao local corretos – por exemplo, a pasta `values-fr` conteria um arquivo `strings.xml` para francês e a pasta `values-es` conteria um arquivo `strings.xml` para espanhol. Você também pode dar nomes a essas pastas com informações regionais – `values-en-rUS` conteria um arquivo `strings.xml` para o inglês norte-americano e `values-en-rGB` conteria um arquivo `strings.xml` para o inglês britânico. Se, para determinada localidade, não forem fornecidos recursos adaptados, o Android usará os recursos *padrão* do aplicativo – isto é, os que estão na subpasta `values` da pasta `res`. Discutiremos essas *convenções de atribuição de nomes de recurso alternativos* com mais detalhes em capítulos posteriores.

Adicionando uma pasta de adaptação local ao projeto do aplicativo

Antes de adicionar uma versão adaptada à localidade do arquivo `strings.xml` do aplicativo **Welcome**, contendo strings em espanhol, você precisa adicionar a pasta `values-es` ao projeto. Para isso:

1. Na janela **Package Explorer** do IDE, clique com o botão direito do mouse na pasta **res** do projeto e selecione **New > Folder** para exibir a caixa de diálogo **New Folder**.
2. No campo **Folder name**: da caixa de diálogo, digite **values-es** e clique em **Finish**.

Esses passos seriam repetidos, com uma pasta **values-localidade** com nome adequado para cada idioma que você quisesse permitir.

Copiando o arquivo strings.xml para a pasta values-es

A seguir, você vai copiar o arquivo **strings.xml** da pasta **values** para a pasta **values-es**. Para isso:

1. Na janela **Package Explorer** do IDE, abra a subpasta **values** da pasta **res** e, em seguida, clique com o botão direito do mouse no arquivo **strings.xml** e selecione **Copy** para copiar o arquivo.
2. Então, clique com o botão direito do mouse na pasta **values-es** e selecione **Paste** para inserir a cópia de **strings.xml** na pasta.

Adaptando as strings para a localidade

Neste aplicativo, a interface gráfica do usuário contém um componente **TextView** que exibe uma string e duas strings de descrição de conteúdo para os componentes **ImageView**. Todas essas strings foram definidas como recursos de string no arquivo **strings.xml**. Agora você pode traduzir as strings na nova versão do arquivo **strings.xml**. As empresas de desenvolvimento de aplicativo frequentemente ou têm tradutores internos ou contratam outras empresas para fazer as traduções. Na verdade, no **Google Play Developer Console** – que você utiliza para publicar seus aplicativos na loja **Google Play** –, é possível encontrar empresas de serviços de tradução. Para obter mais informações sobre o **Google Play Developer Console**, consulte o Capítulo 9 e

developer.android.com/distribute/googleplay/publish/index.html

Para este aplicativo, você vai substituir as strings

```
"Welcome to Android App Development!"
"Android logo"
"Deitel double-thumbs-up bug logo"
```

pelos strings em espanhol

```
"¡Bienvenido al Desarrollo de App Android!"
"Logo de Android"
"El logo de Deitel que tiene el insecto con dedos pulgares hacia arriba"
```

Para isso:

1. Na janela **Package Explorer** do IDE, clique duas vezes no arquivo **strings.xml** da pasta **values-es** a fim de exibir o editor **Android Resources** e, então, selecione o recurso de string **welcome** (Fig. 2.26).
2. No campo **Value**, substitua a string em inglês "Welcome to Android App Development!" pela string em espanhol "¡Bienvenido al Desarrollo de App Android!". Se não puder digitar caracteres e símbolos especiais em espanhol em seu teclado, copie as strings em espanhol de nosso arquivo **res/values-es/strings.xml** na versão final do aplicativo **Welcome** (localizado na pasta **WelcomeInternationalized** com os exemplos do capítulo). Para colar a string em espanhol no campo **Value**, selecione a string em inglês, clique nela com o botão direito do mouse e selecione **Paste**.

3. Em seguida, selecione o recurso `android_logo` e mude seu campo **Value** para "Logo de Android".
4. Por último, selecione o recurso `deitel_logo` e altere seu campo **Value** para "El logo de Deitel que tiene el insecto con dedos pulgares hacia arriba".
5. Exclua os recursos de string `app_name`, `action_settings` e `hello_world` selecionando um por vez e clicando no botão **Remove...**. Será solicitado que você confirme cada operação de exclusão. Esses três recursos foram inseridos no arquivo `strings.xml` padrão quando você criou o projeto do aplicativo. Somente o recurso de string `app_name` é utilizado neste projeto. Vamos explicar o motivo de termos excluído os outros em breve.
6. Salve o arquivo `strings.xml` selecionando **File > Save** ou clicando no ícone  da barra de ferramentas.

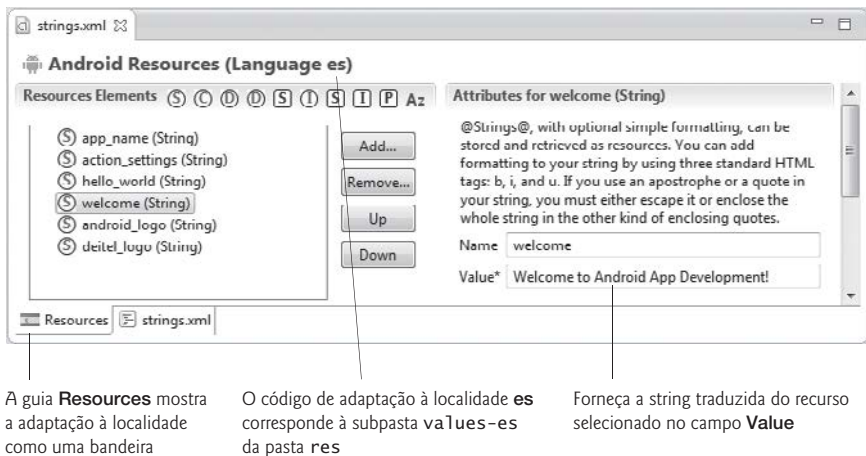


Figura 2.26 Editor **Android Resources** com o recurso de string `welcome` selecionado.

Testando o aplicativo em espanhol

Para testar o aplicativo em espanhol, você precisa alterar as configurações de idioma no emulador do Android (ou em seu dispositivo). Para isso:

1. Toque no ícone Home () no emulador ou em seu dispositivo.
2. Toque no ícone do lançador () e, então, localize e toque no ícone do aplicativo **Settings** ()
3. No aplicativo **Settings**, role até a seção **PERSONAL** e, então, toque em **Language & input**.*
4. Toque em **Language** (o primeiro item da lista) e selecione **Español (España)** na lista de idiomas.

O emulador ou dispositivo muda sua configuração de idioma para espanhol e volta para as configurações de **Language & input**, as quais agora são exibidas em espanhol.

* N. de T. Neste livro, os nomes das opções de menu do Android serão mantidos em inglês porque as traduções não são padronizadas entre os diversos fabricantes.

Em seguida, execute o aplicativo **Welcome** no IDE, o qual instala e executa a versão internacionalizada. A Figura 2.27 mostra o aplicativo executando em espanhol. Quando o aplicativo começa a executar, o Android verifica as configurações de idioma do AVD (ou do dispositivo), determina que o AVD (ou dispositivo) está configurado com espanhol e utiliza os recursos de string `welcome`, `android_logo` e `deitel_logo` definidos em `res/values-es/strings.xml` no aplicativo em execução. Observe, entretanto, que o nome do aplicativo ainda aparece em *inglês* na barra de ação na parte superior do aplicativo. Isso porque *não* fornecemos uma versão adaptada ao local para o recurso de string `app_name` no arquivo `res/values-es/strings.xml`. Lembre-se de que, quando o Android não consegue encontrar uma versão adaptada ao local de um recurso de string, ele usa a versão padrão do arquivo `res/values/strings.xml`.



Erro de programação comum 2.1

Modificar os nomes de recursos pode levar a erros de tempo de execução. O Android usa os nomes de recurso padrão ao carregar recursos adaptados a localidades. Quando criar um arquivo de recurso adaptado ao local, certifique-se de modificar somente os valores dos recursos, não seus nomes.

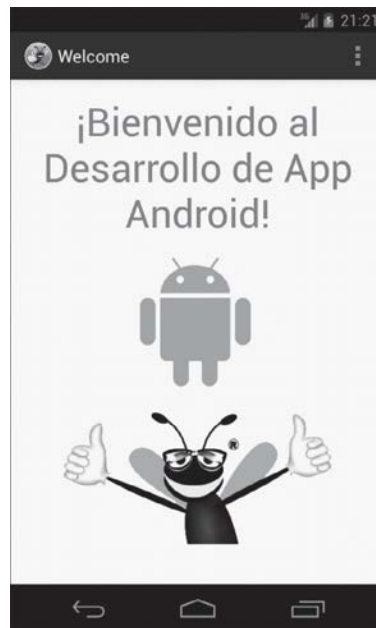


Figura 2.27 Aplicativo **Welcome** executando em espanhol no AVD do Nexus 4.

Retornando o AVD (ou dispositivo) para o inglês

Para retornar o AVD (ou dispositivo) para o inglês:

1. Toque no ícone Home (🏠) no emulador ou em seu dispositivo.
2. Toque no ícone do lançador (⦿) e, então, localize e toque no ícone do aplicativo **Settings** (⚙️) – agora o aplicativo é chamado **Ajustes**, em espanhol.

3. Toque no item **Idioma y entrada de texto** para acessar as configurações de idioma.
4. Toque no item **Idioma** e, então, na lista de idiomas, selecione **English (United States)**.

TalkBack e adaptação ao local

Atualmente, o recurso TalkBack aceita inglês, espanhol, italiano, francês e alemão. Se você executar o aplicativo **Welcome** em um dispositivo com espanhol especificado como idioma e o recurso TalkBack habilitado, o TalkBack pronunciará as strings em espanhol do aplicativo quando cada componente da interface do usuário for tocado.

Na primeira vez que você trocar seu dispositivo para espanhol e habilitar o recurso TalkBack, o Android baixará automaticamente o conversor de texto para voz em espanhol. Se o recurso TalkBack *não* pronunciar as strings em espanhol, então esse conversor ainda não acabou de ser baixado e instalado. Nesse caso, tente executar o aplicativo novamente mais tarde.

Lista de verificação para adaptação ao local

Para obter mais informações sobre a adaptação ao local dos recursos de seu aplicativo, consulte a *Localization Checklist* do Android, em:

developer.android.com/distribute/googleplay/publish/localizing.html

2.9 Para finalizar

Neste capítulo, você usou o IDE Android Developer Tools para construir o aplicativo **Welcome** que exibiu uma mensagem de boas-vindas e duas imagens sem escrever código. Criou uma interface gráfica de usuário simples usando o editor **Graphical Layout** do IDE e configurou propriedades dos componentes da interface usando a janela **Properties**.

O aplicativo apresentou texto em um componente **TextView** e imagens em componentes **ImageView**. Você modificou o componente **TextView** da interface de usuário padrão para exibir o texto do aplicativo centralizado na interface, com um tamanho de fonte maior e em uma das cores de tema padrão. Também usou o componente **Palette** dos controles da interface do usuário do editor **Graphical Layout** para arrastar e soltar componentes **ImageView** na interface. Seguindo as melhores práticas, você definiu todas as strings e valores numéricos em arquivos de recurso na pasta **res** do projeto.

Aprendeu que o Android possui recursos de acessibilidade para ajudar pessoas com vários tipos de deficiências a usar seus dispositivos. Mostramos como habilitar o recurso TalkBack do Android para permitir que um dispositivo pronuncie o texto da tela ou o texto que você fornece a fim de ajudar o usuário a entender o objetivo e o conteúdo de um componente da interface gráfica. Discutimos o recurso **Explore by Touch** do Android, o qual permite ao usuário tocar na tela para ouvir o aplicativo TalkBack falar o que está na tela próximo ao local do toque. Para os componentes **ImageView** do aplicativo, você forneceu descrições do conteúdo que podem ser usadas com TalkBack e **Explore by Touch**.

Por fim, você aprendeu a usar os recursos de internacionalização do Android para atingir o maior público possível para seus aplicativos. Você adaptou o aplicativo **Welcome** para a localidade com strings em espanhol para o texto do componente **TextView** e strings de acessibilidade dos componentes **ImageView**, e testou o aplicativo em um AVD configurado para espanhol.

O desenvolvimento com Android é uma combinação de projeto de interface gráfica do usuário e codificação em Java. No próximo capítulo, você vai desenvolver um aplicativo simples para calcular gorjetas, chamado de **TIP Calculator**, utilizando o editor **Graphical Layout** para criar a interface gráfica do usuário visualmente e utilizando a programação com Java para especificar o comportamento do aplicativo.

Exercícios de revisão

- 2.1** Preencha os espaços em branco em cada um dos seguintes enunciados:
- a) Os arquivos de layout são considerados como recursos do aplicativo e são armazenados na pasta _____ do projeto. Os layouts de interface gráfica do usuário são colocados dentro da subpasta layout dessa pasta.
 - b) Ao projetar uma interface gráfica do usuário no Android, você normalmente deseja que ela _____ para que apareça corretamente em vários dispositivos.
 - c) Você pode _____ facilmente seu aplicativo para a localidade criando arquivos de recurso XML adicionais para recursos de string em outros idiomas.
 - d) As duas unidades de medida para pixels independentes de densidade são _____ e _____.
 - e) Para executar um aplicativo em um AVD (Dispositivo Android Virtual), clique com o botão direito do mouse no nó-raiz do aplicativo no Eclipse na janela _____ e selecione **Run As > Android Application**.
 - f) O _____ permite ao usuário ouvir o aplicativo TalkBack pronunciar o que está no local onde o usuário toca na tela.
 - g) O Android usa um esquema de atribuição de nomes de pasta especial para escolher automaticamente os recursos adaptados ao local corretos — por exemplo, a pasta _____ conteria um arquivo `strings.xml` para francês e a pasta _____ conteria um arquivo `strings.xml` para espanhol.
- 2.2** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- a) O IDE Android Development Tools é o mais usado para criar e testar aplicativos Android.
 - b) O layout `RelativeLayout` organiza os componentes um em relação ao outro ou em relação ao seu contêiner pai.
 - c) Para centralizar o texto no componente `TextView`, configure sua propriedade `Alignment` como `center`.
 - d) O Android contém recursos de acessibilidade para ajudar pessoas com vários tipos de deficiência a usar seus dispositivos.
 - e) Para deficientes visuais, o recurso `SpeakBack` do Android pode pronunciar o texto que está na tela ou o que você fornece para ajudar o usuário a entender a finalidade de um componente da interface.
 - f) Considera-se uma melhor prática no Android garantir que todo componente de interface gráfica do usuário possa ser usado com o recurso `TalkBack`, fornecendo-se texto para a propriedade `Content Description` de todo componente que não exiba texto.

Respostas dos exercícios de revisão

- 2.1** a) res. b) possa mudar de escala. c) adaptar. d) `dp` e `dip`. e) **Project Explorer**. f) **Explore by Touch**. g) `values-fr`, `values-es`.

- 2.2** a) Verdadeira. b) Verdadeira. c) Falsa. Para centralizar o texto no componente `TextView`, configure sua propriedade `Gravity` como `center`. d) Verdadeira. e) Falsa. O recurso se chama `TalkBack`. f) Verdadeira.

Exercícios

- 2.3** Preencha os espaços em branco em cada um dos seguintes enunciados:
- a) O _____ do ADT permite construir interfaces gráficas utilizando técnicas de arrastar e soltar.
 - b) Para um aplicativo Android criado com o Eclipse, o layout da interface gráfica é armazenado em um arquivo XML chamado _____, por padrão.
 - c) A interface gráfica do usuário padrão de um aplicativo **Blank Page** consiste em um _____ (layout) com um fundo cinza claro e um componente `TextView` contendo "Hello World!".
 - d) A pasta `res` de seu projeto contém três subpastas para imagens – `drawable-hdpi` (densidade alta), `drawable-mdpi` (densidade média) e `drawable-ldpi` (densidade baixa). Essas pastas armazenam imagens com diferentes densidades _____.
 - e) A documentação para suporte de vários tamanhos de tela recomenda usar pixels independentes de densidade para as dimensões dos componentes da interface gráfica do usuário e outros elementos de tela, e _____ para tamanhos de fonte.
 - f) Um pixel independente de densidade é equivalente a um pixel em uma tela com 160 dpi (pontos por polegada). Em uma tela com 240 dpi, cada pixel independente de densidade vai mudar de escala por um fator de _____.
 - g) Em uma tela com 120 dpi, cada pixel independente de densidade vai mudar de escala por um fator de _____. Assim, o mesmo componente com 100 pixels independentes de densidade de largura vai ter 75 pixels de largura reais.
- 2.4** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- a) Para que as imagens sejam bem adaptadas, um dispositivo de alta densidade de pixels precisa de imagens de resolução mais baixa do que um dispositivo de baixa densidade de pixels.
 - b) É considerada uma boa prática "externalizar" strings, arrays de string, imagens, cores, tamanhos de fonte, dimensões e outros recursos de aplicativo para que você ou outra pessoa de sua equipe possa gerenciá-los separadamente do código de seu aplicativo.
 - c) Você pode usar o editor **Visual Layout** para criar um aplicativo Android sem escrever código algum.
- 2.5** (**Aplicativo Scrapbooking**) Encontre quatro imagens de código-fonte aberto de pontos de referência famosos utilizando sites como o Flickr. Crie um aplicativo no qual você organize as imagens em uma colagem. Adicione texto identificando cada ponto de referência. Lembre-se de que os nomes de imagem devem utilizar apenas letras minúsculas.
- 2.6** (**Aplicativo Scrapbooking com acessibilidade**) Usando as técnicas aprendidas na Seção 2.7, aprimore sua solução para o Exercício 2.5, fornecendo strings que possam ser usadas com o recurso de acessibilidade `TalkBack` do Android. Caso você disponha de um dispositivo Android, teste o aplicativo nesse dispositivo com o recurso `TalkBack` habilitado.
- 2.7** (**Aplicativo Scrapbooking com internacionalização**) Usando as técnicas aprendidas na Seção 2.8, aprimore sua solução para o Exercício 2.6, definindo um conjunto de strings para outro idioma falado. Use um serviço de tradutor online, com o `translate.google.com`, para traduzir as strings e coloque-as no arquivo de recurso `strings.xml` apropriado. Use as instruções da Seção 2.8 para testar o aplicativo em um AVD (ou em um dispositivo, caso disponha de um).

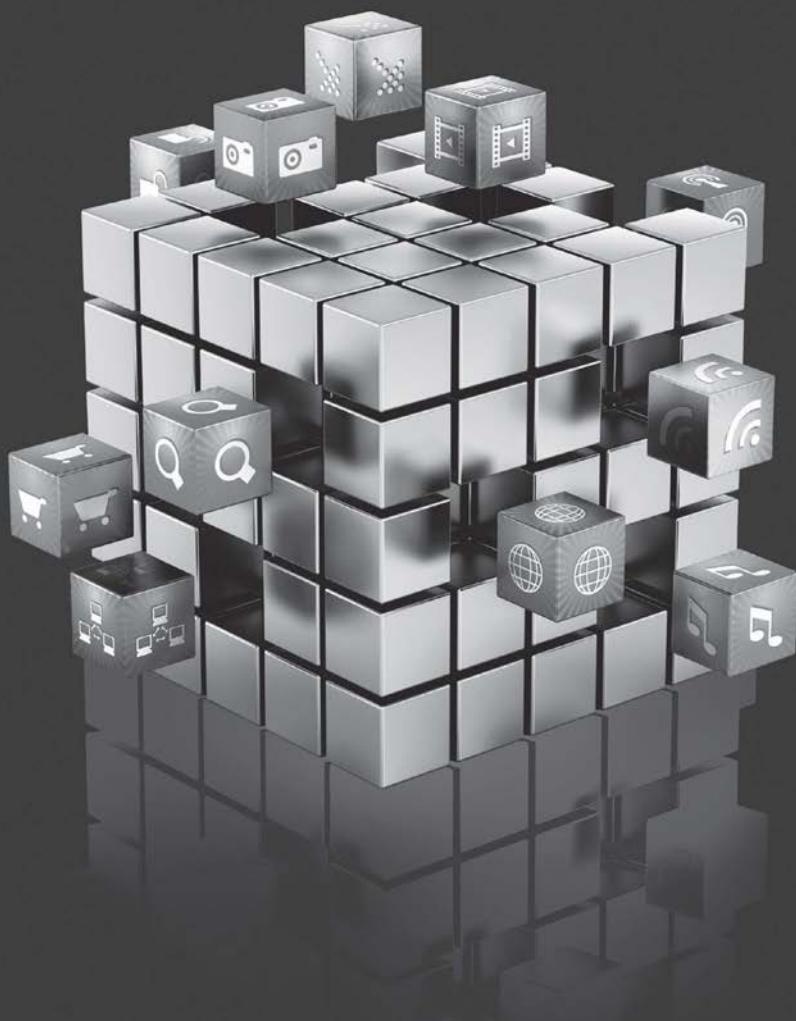
3

Aplicativo Tip Calculator

Objetivos

Neste capítulo, você vai:

- Projetar uma interface gráfica do usuário usando componentes `LinearLayout` e `GridLayout`.
- Usar a janela **Outline** do IDE para adicionar elementos de interface gráfica do usuário a componentes `LinearLayout` e `GridLayout`.
- Usar componentes de interface gráfica do usuário `TextView`, `EditText` e `SeekBar`.
- Usar recursos de programação orientada a objetos com Java, incluindo classes, objetos, interfaces, classes internas anônimas e herança para adicionar funcionalidades a um aplicativo Android.
- Interagir com elementos da interface gráfica do usuário via programação para alterar o texto que eles exibem.
- Usar tratamento de eventos para responder às interações do usuário com componentes `EditText` e `SeekBar`.
- Especificar que o teclado numérico sempre deve aparecer quando um aplicativo estiver executando.
- Especificar que um aplicativo suporta apenas orientação retrato.



3.1	Introdução	3.3.7	Implementação da interface
3.2	Teste do aplicativo Tip Calculator		<code>OnSeekBarChangeListener</code> para
3.3	Visão geral das tecnologias		lidar com alterações na posição do
	3.3.1 Classe <code>Activity</code>		cursor no componente <code>SeekBar</code>
	3.3.2 Métodos de ciclo de vida de	3.3.8	<code>AndroidManifest.xml</code>
	<code>Activity</code>	3.4	Construção da interface gráfica do
	3.3.3 Organização de componentes de		usuário do aplicativo
	visualização com <code>GridLayout</code> e	3.4.1	Introdução ao componente
	<code>LinearLayout</code>		<code>GridLayout</code>
	3.3.4 Criação e personalização da interface	3.4.2	Criação do projeto <code>TipCalculator</code>
	gráfica do usuário com o editor	3.4.3	Alteração para um componente
	Graphical Layout e com as janelas		<code>GridLayout</code>
	Outline e Properties	3.4.4	Adição dos componentes <code>TextView</code> ,
	3.3.5 Formatação de números como moeda		<code>EditText</code> e <code>LinearLayout</code>
	corrente específica da localidade e	3.4.5	Personalização das visualizações para
	strings de porcentagem		concluir o projeto
	3.3.6 Implementação da interface	3.5	Adição de funcionalidade ao aplicativo
	<code>TextWatcher</code> para lidar com	3.6	<code>AndroidManifest.xml</code>
	alterações de texto em componente	3.7	Para finalizar
	<code>EditText</code>		

Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios

3.1 Introdução

O aplicativo **Tip Calculator** (Fig. 3.1(a)) calcula e exibe possíveis gorjetas para contas de um restaurante. À medida que você insere cada dígito do valor de uma conta, tocando

a) Interface gráfica inicial do usuário



Mova o cursor do componente **SeekBar** para alterar a porcentagem de gorjeta personalizada

A porcentagem de gorjeta personalizada selecionada com o componente **SeekBar** aparece aqui

Use o teclado numérico para inserir o valor da conta com centavos – o aplicativo dividirá o valor inserido por 100.0 para calcular o valor da conta

Use o botão de exclusão  para remover dígitos da direita para a esquerda

b) Interface gráfica do usuário depois que ele insere o valor 34.56 e muda a porcentagem de gorjeta personalizada para 20%



Figura 3.1 Inserindo o total da conta e calculando a gorjeta.

no *teclado numérico*, o aplicativo calcula e exibe o valor da gorjeta e o total da conta (valor da conta + gorjeta) para uma gorjeta de 15% e para uma porcentagem de gorjeta personalizada (18% por padrão). Você pode especificar uma porcentagem de gorjeta personalizada de 0% a 30%, movendo o *cursor* de um componente *SeekBar* – isso atualiza a porcentagem personalizada mostrada e exibe a gorjeta personalizada e o total (Fig. 3.1(b)). Escolhemos 18% como porcentagem personalizada padrão porque muitos restaurantes nos Estados Unidos acrescentam essa taxa de serviço para festas com seis pessoas ou mais. O teclado numérico na Fig. 3.1 pode ser diferente de acordo com a versão de Android de seu AVD ou dispositivo ou se você tiver instalado e selecionado um teclado personalizado em seu dispositivo.

Começaremos testando o aplicativo – você vai utilizá-lo para calcular gorjetas de 15% e personalizadas. Em seguida, daremos uma visão geral das tecnologias utilizadas para criar o aplicativo. Você vai construir a interface gráfica do aplicativo usando o editor *Graphical Layout* do IDE Android Developer Tools e a janela *Outline*. Por fim, vamos apresentar o código Java completo do aplicativo e fazer um acompanhamento detalhado. Fornecemos online uma versão para Android Studio das Seções 3.2 e 3.4 (em inglês) em <http://www.deitel.com/books/AndroidHTP2>.

3.2 Teste do aplicativo Tip Calculator

Abra e execute o aplicativo

Abra o IDE Android Developer Tools e importe o projeto do aplicativo *Tip Calculator*. Execute os passos a seguir:

1. **Ative o AVD do Nexus 4.** Para este teste, usaremos o AVD do smartphone Nexus 4 que você configurou na seção “Antes de começar”. Para ativar o AVD do Nexus 4, selecione *Window > Android Virtual Device Manager* a fim de exibir a caixa de diálogo *Android Virtual Device Manager*. Selecione o AVD do Nexus 4 e clique em *Start...*; em seguida, clique no botão *Launch* na caixa de diálogo *Launch Options* que aparece.
2. **Abra a caixa de diálogo *Import*.** Selecione *File > Import...* para abrir a caixa de diálogo *Import*.
3. **Importe o projeto do aplicativo *Tip Calculator*.** Expanda o nó *General*, selecione *Existing Projects into Workspace* e, em seguida, clique em *Next >* para ir ao passo *Import Projects*. Certifique-se de que *Select root directory* esteja selecionado e, em seguida, clique em *Browse....* Na caixa de diálogo *Browse For Folder*, localize a pasta *TipCalculator* na pasta de exemplos do livro, selecione-a e clique em *OK*. Certifique-se de que *Copy projects into workspace* *não* esteja selecionado. Clique em *Finish* para importar o projeto. Agora ele aparece na janela *Package Explorer*.
4. **Ative o aplicativo *Tip Calculator*.** Clique com o botão direito do mouse no projeto *TipCalculator* na janela *Package Explorer* e selecione *Run As > Android Application* para executar o aplicativo *Tip Calculator* no AVD.

Digite um total de conta

Usando o teclado numérico, digite 34.56. Basta digitar 3456 – o aplicativo posicionará os centavos à direita do ponto decimal. Se cometer um erro, pressione o botão de exclusão () para apagar um dígito por vez. Os componentes *TextView* sob os rótulos da gorjeta de 15% e da porcentagem de gorjeta personalizada (18% por padrão) mostram o valor da gorjeta e o total da conta para essas porcentagens. Todos os componentes *TextView* *Tip* e *Total* são atualizados sempre que você insere ou exclui um dígito.

Selecione uma porcentagem de gorjeta personalizada

Use o componente Seekbar para especificar uma porcentagem de gorjeta *personalizada*. Arraste o *cursor* do componente Seekbar até que a porcentagem personalizada indique 20% (Fig. 3.1(b)). À medida que você arrasta o cursor, a gorjeta e o total para essa porcentagem de gorjeta personalizada são atualizados continuamente. Por padrão, o componente Seekbar permite selecionar valores de 0 a 100, mas especificamos um valor máximo de 30 para este aplicativo.

3.3 Visão geral das tecnologias

Esta seção apresenta os recursos do IDE e as tecnologias do Android que você vai usar para construir o aplicativo **Tip Calculator**. Supomos que você *já* conhece programação orientada a objetos com Java – se não conhece, os apêndices contêm uma introdução a Java. Você vai:

- Usar várias classes Android para criar objetos.
- Chamar métodos em classes e objetos Android.
- Definir e chamar seus próprios métodos.
- Usar herança para criar uma subclasse da classe `Activity` do Android que define as funcionalidades do aplicativo **Tip Calculator**.
- Usar tratamento de eventos, classes internas anônimas e interfaces para processar as interações da interface gráfica do usuário.

3.3.1 Classe Activity

Ao contrário de muitos aplicativos Java, os aplicativos Android *não* têm um *método main*. Em vez disso, eles têm quatro tipos de componentes executáveis – *atividades, serviços, provedores de conteúdo e receptores de transmissão por broadcast*. Neste capítulo, vamos discutir as atividades, as quais são definidas como subclasses de `Activity` (pacote `android.app`). Os usuários interagem com uma atividade por meio de *componentes de visualização** – isto é, componentes da interface gráfica do usuário. Antes do Android 3.0, normalmente uma atividade distinta era associada a cada tela de um aplicativo. Como você vai ver a partir do Capítulo 5, uma atividade pode gerenciar vários fragmentos. Em um telefone, cada fragmento geralmente ocupa a tela inteira e a atividade alterna entre os fragmentos com base nas interações do usuário. Em um tablet, as atividades frequentemente exibem vários fragmentos por tela para aproveitar melhor o tamanho grande da tela.

3.3.2 Métodos de ciclo de vida de Activity

Ao longo de toda sua vida, uma atividade pode estar em um dentre vários *estados* – *ativa* (isto é, *em execução*), *pausada* ou *parada*. A atividade transita entre esses estados em resposta a vários *eventos*.

- Uma atividade *ativa* é *visível* na tela e “tem o foco” – isto é, está no *primeiro plano*. Essa é a atividade com que o usuário está interagindo.
- Uma atividade *pausada* é *visível* na tela, mas *não* tem o foco – como quando uma caixa de diálogo de alerta é exibida.

* N. de T. Neste livro, os termos *visualização*, *Views* e *componentes de visualização* são usados de forma intercambiada, conforme o contexto. Todos referem-se a componentes do tipo `View`.

- Uma atividade *parada não é visível* na tela e é provável que seja encerrada pelo sistema quando a memória que ocupa for necessária. Uma atividade é *parada* quando outra se torna *ativa*.

À medida que uma atividade transita entre esses estados, o runtime do Android chama vários *métodos de ciclo de vida* – todos os quais são definidos na classe `Activity`

<http://developer.android.com/reference/android/app/Activity.html>

Você vai sobrescrever o método `onCreate` em *cada* atividade. Esse método é chamado pelo runtime do Android quando uma atividade está *começando* – isto é, quando sua interface gráfica do usuário está prestes a ser exibida para que o usuário possa interagir com a atividade. Outros métodos de ciclo de vida importantes incluem `onStart`, `onPause`, `onRestart`, `onResume`, `onStop` e `onDestroy`. Vamos discutir a *maioria* desses métodos em capítulos posteriores. Cada método de ciclo de vida de atividade que você sobrescreve *deve* chamar a versão da superclasse; caso contrário, ocorrerá uma *exceção*. Isso é necessário porque cada método de ciclo de vida na superclasse `Activity` contém o código que deve ser executado, além do código que você define em seus métodos de ciclo de vida sobrescritos.

3.3.3 Organização de componentes de visualização com `LinearLayout` e `GridLayout`

Lembre-se de que os layouts organizam os componentes de visualização em uma interface gráfica do usuário. Um componente `LinearLayout` (pacote `android.widget`) organiza os views *horizontalmente* (o padrão) ou *verticalmente* e pode dimensioná-los proporcionalmente. Usaremos isso para organizar dois componentes `TextView` horizontalmente e garantir que cada um utilize metade do espaço horizontal disponível.

`GridLayout` (pacote `android.widget`) foi introduzido no Android 4.0 como um novo layout para organizar views em células em uma grade retangular. As células podem ocupar *várias* linhas e colunas, possibilitando layouts complexos. Em muitos casos, `GridLayout` pode ser usado para substituir o componente `TableLayout`, mais antigo e às vezes menos eficiente, que organiza os componentes de visualização em linhas e colunas, em que cada linha normalmente é definida como um componente `TableRow` e o número de colunas é definido pelo componente `TableRow` que contém a maioria das células. Normalmente, `GridLayout` exige nível de API 14 ou mais alto. Contudo, a *Android Support Library* fornece versões alternativas de `GridLayout` e de muitos outros recursos de interface gráfica do usuário para que você possa utilizá-los em versões mais antigas do Android. Para obter mais informações sobre essa biblioteca e como utilizá-la em seus aplicativos, visite:

<http://developer.android.com/tools/support-library/index.html>

Um componente `GridLayout` *não pode* especificar, dentro de determinada linha, que o espaço horizontal deve ser alocado *proporcionalmente* entre vários componentes de visualização. Por isso, várias linhas na interface gráfica do usuário deste aplicativo colocarão dois componentes `TextView` em um componente `LinearLayout` horizontal. Isso permitirá colocar dois componentes `TextView` na mesma célula de `GridLayout` e dividir o espaço da célula igualmente entre eles. Vamos abordar mais layouts e visualizações em capítulos posteriores – para ver uma lista completa, visite:

<http://developer.android.com/reference/android/widget/package-summary.html>

3.3.4 Criação e personalização da interface gráfica do usuário com o editor Graphical Layout e com as janelas Outline e Properties

Você vai criar componentes `TextView`, `EditText` e `SeekBar` usando o editor **Graphical Layout** do IDE (que foi utilizado no Capítulo 2) e a janela **Outline**; então, vai personalizá-los com a janela **Properties** do IDE – a qual aparece na parte inferior da janela **Outline** quando se está editando uma interface gráfica no editor **Graphical Layout**. Isso vai ser feito *sem* manipular diretamente o código XML armazenado nos arquivos da pasta `res` do projeto.

Um componente **EditText** – frequentemente chamado de *caixa de texto* ou *campo de texto* em outras tecnologias de interface gráfica do usuário – é uma *subclasse* de `TextView` (apresentado no Capítulo 2) que pode exibir texto e aceitar entrada de texto do usuário. Você vai especificar um componente `EditText` para entrada *numérica*, só vai permitir que os usuários insiram dígitos e vai restringir o número *máximo* de dígitos que podem ser inseridos.

Um componente **SeekBar** – frequentemente chamado de *controle deslizante* em outras tecnologias de interface gráfica do usuário – representa um valor inteiro no intervalo 0 a 100 por padrão e permite ao usuário selecionar um número nesse intervalo movendo o cursor do componente `SeekBar`. Você vai personalizar o componente `SeekBar` de modo que o usuário possa escolher uma porcentagem de gorjeta personalizada *somente* no intervalo mais limitado de 0 a 30.

Na janela **Properties**, as propriedades mais comumente personalizadas de uma visualização normalmente aparecem na parte superior, com seus nomes exibidos em negrito (Fig. 3.2). Todas as propriedades de uma visualização também são organizadas em categorias dentro da janela **Properties**. Por exemplo, a classe `TextView` herda muitas propriedades da classe `View`; portanto, a janela **Properties** exibe uma categoria **TextView** com propriedades específicas para esse componente, seguida de uma categoria **View** com propriedades herdadas da classe `View`.

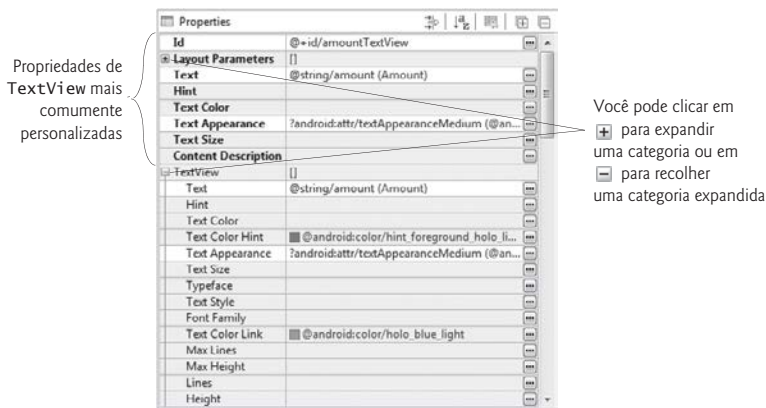


Figura 3.2 Janela **Properties** mostrando as propriedades mais comumente personalizadas de um componente `TextView`.

3.3.5 Formatação de números como moeda corrente específica da localidade e strings de porcentagem

Você vai usar a classe `NumberFormat` (pacote `java.text`) para criar moeda corrente *específica da localidade* e strings de porcentagem – uma parte importante da *internacionalização*. Você

também poderia adicionar strings de *acessibilidade* e internacionalizar o aplicativo usando as técnicas que aprendeu nas Seções 2.7 e 2.8, embora não tenhamos feito isso neste aplicativo.

3.3.6 Implementação da interface `TextWatcher` para lidar com alterações de texto em componente `EditText`

Você vai usar uma *classe interna anônima* para implementar a interface `TextWatcher` (do pacote `android.text`) para responder aos *eventos gerados quando o usuário altera o texto* no componente `EditText` deste aplicativo. Em particular, vai usar o método `onTextChanged` para exibir o valor da conta formatado em moeda corrente e para calcular a gorjeta e o total à medida que o usuário insere cada dígito.

3.3.7 Implementação da interface `OnSeekBarChangeListener` para lidar com alterações na posição do cursor no componente `SeekBar`

Você vai implementar a interface `SeekBar.OnSeekBarChangeListener` (do pacote `android.widget`) para responder ao movimento do *cursor* do componente `SeekBar` feito pelo usuário. Em particular, vai usar o método `onProgressChanged` para exibir a porcentagem de gorjeta personalizada e para calcular a gorjeta e o total à medida que o usuário move o cursor do componente `SeekBar`.

3.3.8 `AndroidManifest.xml`

O arquivo `AndroidManifest.xml` é gerado pelo IDE quando um novo projeto de aplicativo é criado. Esse arquivo contém muitas das configurações que você especifica na caixa de diálogo `New Android Application`, como o nome do aplicativo, o nome do pacote, SDKs alvo e mínimo, nome(s) de atividade, tema e muito mais. Você vai usar o editor `Android Manifest` do IDE para adicionar ao manifesto uma nova configuração que obriga o *teclado virtual* a permanecer na tela. Também vai especificar que o aplicativo aceita apenas *orientação retrato* – isto é, o lado maior do dispositivo é vertical.

3.4 Construção da interface gráfica do usuário do aplicativo

Nesta seção, mostraremos os passos exatos para construir a interface gráfica do usuário do aplicativo `Tip Calculator`. A interface gráfica do usuário não será parecida com a mostrada na Fig. 3.1 até que você tenha concluído os passos. À medida que prosseguir nesta seção, o número de detalhes apresentados poderá parecer grande, mas eles são repetitivos e você se acostumará com eles conforme usar o IDE.

3.4.1 Introdução ao componente `GridLayout`

Este aplicativo utiliza um componente `GridLayout` (Fig. 3.3) para organizar os componentes de visualização em cinco *linhas* e duas *colunas*. Cada célula em um componente `GridLayout` pode estar *vazia* ou conter uma ou mais *visualizações* (componentes do tipo *view*), incluindo layouts *contendo* outras visualizações. Essas *views* podem abranger *várias* linhas ou colunas, embora não tenhamos usado esse recurso nesta interface de usuário. O número de linhas e colunas de um componente `GridLayout` é especificado na janela `Properties`.

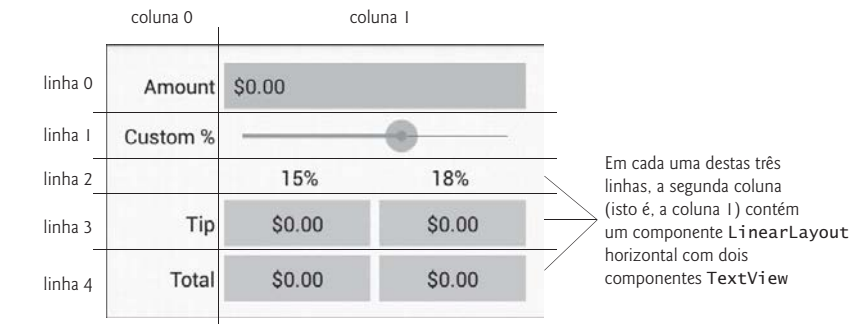


Figura 3.3 Componente `GridLayout` da interface gráfica do usuário do aplicativo **Tip Calculator** rotulado por suas linhas e colunas.

A *altura* de cada linha é determinada pelo componente *view* *mais alto* nessa linha. Da mesma forma, a *largura* de uma coluna é definida pelo componente *view* *mais largo* nessa coluna. Por padrão, os componentes são adicionados da esquerda para a direita em uma linha. Conforme você verá, é possível especificar a linha e coluna exatas nas quais um componente *view* deve ser colocado. Vamos discutir outros recursos de `GridLayout` quando apresentarmos os passos para a construção da interface gráfica do usuário. Para saber mais sobre a classe `GridLayout`, visite:

<http://developer.android.com/reference/android/widget/GridLayout.html>

Valores da propriedade `Id` para as visualizações deste aplicativo

A Figura 3.4 mostra os valores da propriedade `Id` das *views*. Por clareza, nossa convenção de atribuição de nomes é usar o nome da classe da *view* na propriedade `Id` da visualização e no nome da variável Java.

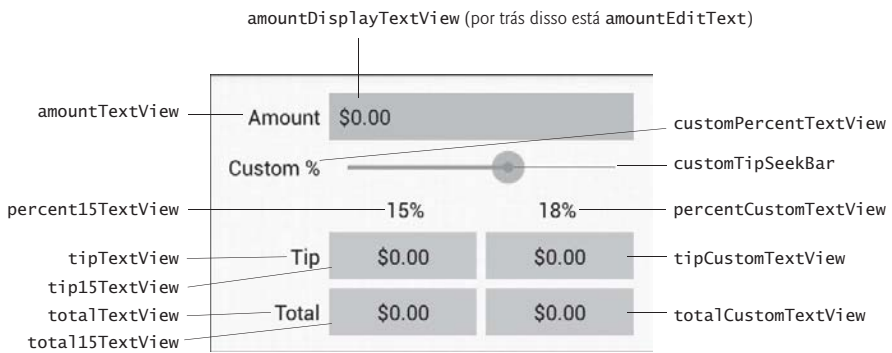


Figura 3.4 Componentes da interface gráfica do usuário do aplicativo **Tip Calculator** rotulados com seus valores de propriedade `Id`.

Na coluna da direita da primeira linha, existem, na verdade, *dois* componentes na *mesma* célula da grade – o componente `amountDisplayTextView` está *ocultando* o componente `amountEditText` que recebe a entrada do usuário. Conforme você verá em breve, restringimos a entrada do usuário a dígitos inteiros para que ele não possa inserir entrada inválida. Contudo, queremos que ele veja o valor da conta como *moeda corrente*. À medida que o usuário insere cada dígito, dividimos o valor por 100,0 e exibimos o resultado

formatado em moeda corrente no componente `amountDisplayTextView`. No *local U.S.*, se o usuário digitar 3456, à medida que cada dígito for inserido, o componente `amountDisplayTextView` mostrará os valores \$0.03, \$0.34, \$3.45 e \$34.56, respectivamente.

Valores da propriedade `Id` de `LinearLayout`

A Figura 3.5 mostra os valores de `Id` dos três componentes `LinearLayout` horizontais na coluna da direita de `GridLayout`.

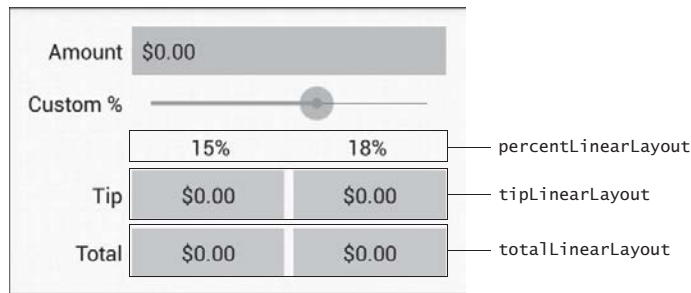


Figura 3.5 Componentes `LinearLayout` da interface gráfica do usuário do aplicativo **Tip Calculator** com seus valores de propriedade `Id`.

3.4.2 Criação do projeto **TipCalculator**

O IDE Android Developer Tools só permite *um* projeto com determinado nome por área de trabalho; portanto, antes de criar o novo projeto, exclua o projeto **TipCalculator** que você testou na Seção 3.2. Para isso, clique nele com o botão direito do mouse e selecione **Delete**. Na caixa de diálogo que aparece, certifique-se de que **Delete project contents on disk** *não* esteja selecionado e, em seguida, clique em **OK**. Isso remove o projeto da área de trabalho, mas deixa a pasta e os arquivos do projeto no disco para o caso de, posteriormente, você querer ver nosso aplicativo original outra vez.

Criando um novo projeto **Blank App**

Crie um novo **Android Application Project**. Especifique os seguintes valores no primeiro passo de **New Android Application** da caixa de diálogo **New Android Project** e, em seguida, pressione **Next >**:

- **Application Name:** Tip Calculator
- **Project Name:** TipCalculator
- **Package Name:** com.deitel.tipcalculator
- **Minimum Required SDK:** API18: Android 4.3
- **Target SDK:** API19: Android 4.4
- **Compile With:** API19: Android 4.4
- **Theme:** Holo Light with Dark Action Bar
- **Create Activity:** TipCalculator
- **Build Target:** certifique-se de que **Android 4.3** esteja marcado

No segundo passo de **New Android Application** da caixa de diálogo **New Android Project**, deixe as configurações padrão e pressione **Next >**. No passo **Configure Launcher Icon**, clique no botão **Browse...**, selecione a imagem de ícone de aplicativo **DeitelGreen.png** (fornecida

na pasta `images` com os exemplos do livro), clique no botão **Open** e pressione **Next >**. No passo **Create Activity**, selecione **Blank Activity** (mantenha o nome de atividade padrão) e pressione **Next >**. No passo **Blank Activity**, deixe as configurações padrão e pressione **Finish** para criar o projeto. No editor **Graphical Layout**, selecione **Nexus 4** na lista suspensa de tipo de tela (como na Fig. 2.12). Mais uma vez, usaremos esse dispositivo como base para nosso projeto.

3.4.3 Alteração para um componente GridLayout

O layout padrão em `activity_main.xml` é `FrameLayout`. Aqui, você vai mudar isso para `GridLayout`. Na janela **Outline**, clique com o botão direito do mouse em `FrameLayout` e selecione **ChangeLayout...**. Na caixa de diálogo **ChangeLayout**, selecione `GridLayout` e clique em **OK**. O IDE muda o layout e configura sua propriedade `Id` como `GridLayout1`. Mudamos isso para `GridLayout` usando o campo `Id` na janela **Properties**. Por padrão, a propriedade `Orientation` de `GridLayout` está configurada como horizontal, indicando que seu conteúdo será disposto linha por linha. Configure as propriedades `Padding Left` e `Padding Right` de `GridLayout` como `activity_horizontal_margin` e `Padding Top` e `Padding Bottom` como `activity_vertical_margin`.

Especificando duas colunas e margens padrão para o componente GridLayout

Lembre-se de que a interface gráfica do usuário na Fig. 3.3 consiste em duas colunas. Para especificar isso, selecione `GridLayout` na janela **Outline** e, então, mude sua propriedade `Column Count` para 2 (no grupo `GridLayout` da janela **Properties**). Por padrão, *não há uma margem* – espaços que separam visualizações – em torno das células de um componente `GridLayout`. Configure a propriedade `Use Default Margins` do componente `GridLayout` como `true` para indicar que ele deve colocar margens em torno de suas células. Por padrão, o componente `GridLayout` usa a distância recomendada entre visualizações (8dp), conforme especificado em

<http://developer.android.com/design/style/metrics-grids.html>

3.4.4 Adição dos componentes TextView, EditText, SeekBar e LinearLayout

Agora vamos à construção da interface gráfica do usuário da Figura 3.3. Você vai começar com o layout e as visualizações (views) básicas nesta seção. Na Seção 3.4.5, você vai personalizar as propriedades das visualizações (views) para concluir o projeto. À medida que adicionar cada visualização à interface do usuário, configure sua propriedade `Id` imediatamente, usando os nomes que aparecem nas Figs. 3.4 e 3.5. A propriedade `Id` da visualização selecionada pode ser alterada na janela **Properties** ou clicando com o botão direito do mouse na visualização (no editor **Graphical Layout** ou na janela **Outline**), selecionando **Edit ID...** e alterando a propriedade `Id` na caixa de diálogo **Rename Resource** que aparece.

Nos passos a seguir, você vai usar a janela **Outline** para adicionar visualizações ao componente `GridLayout`. Ao trabalhar com layouts, pode ser difícil ver suas *estruturas aninhadas* e colocar visualizações nos locais corretos, arrastando-as para a janela do editor **Graphical Layout**. A janela **Outline** torna essas tarefas mais fáceis, pois mostra a estrutura aninhada da interface gráfica do usuário. Execute os passos a seguir exatamente na ordem especificada – caso contrário, as views *não* aparecerão na ordem correta em cada linha. Se isso acontecer, você pode reordenar as visualizações arrastando-as na janela **Outline**.

Passo 1: Adicionando visualizações à primeira linha

A primeira linha é constituída do componente `amountTextView` na primeira coluna e do componente `amountEditText` atrás de `amountDisplayTextView` na segunda coluna. Sempre que você solta uma *view* ou layout no componente `gridLayout` na janela **Outline**, a *view* é colocada na *próxima célula aberta* do layout, a não ser que especifique de forma diferente, configurando suas propriedades **Row** e **Column**. Você vai fazer isso neste passo para que `amountEditText` e `amountDisplayTextView` sejam inseridos na mesma célula.

Todos os componentes `TextView` deste aplicativo utilizam a fonte de tamanho *médio* do tema do aplicativo. A **Palette** do editor **Graphical Layout** fornece componentes `TextView` *previamente configurados*, chamados **Large**, **Medium** e **Small** (na seção **Form Widgets**), para representar os tamanhos de texto correspondentes do tema. Em cada caso, o IDE configura a propriedade **Text Appearance** do componente `TextView` de forma correspondente. Execute as tarefas a seguir para adicionar os dois componentes `TextView` e o componente `EditText`:

1. Arraste um componente `TextView` **Medium** da seção **Form Widgets** da **Palette** e solte-o no componente `gridLayout` na janela **Outline**. O IDE cria um novo componente `TextView` chamado `textView1` e o aninha no nó de `gridLayout`. O texto padrão "Medium Text" aparece no editor **Graphical Layout**. Altere a propriedade **Id** de `TextView` para `amountTextView`. Você vai alterar seu texto no passo 6 (Seção 3.4.5).
2. Este aplicativo permite inserir somente valores *inteiros não negativos*, os quais ele divide por 100.0 para exibir o valor da conta. A seção **Text Fields** da **Palette** fornece muitos componentes `EditText` *previamente configurados* para várias formas de entrada (por exemplo, números, horas, datas, endereços e números de telefone). Quando o usuário interage com um componente `EditText`, é exibido um teclado apropriado, com base no *tipo de entrada* do componente. Quando você deixa o cursor alguns instantes sobre um componente `EditText` na **Palette**, uma *dica de ferramenta* indica o tipo de entrada. Na seção **Text Fields** da **Palette**, arraste um componente `EditText` **Number** (exibido com o número 42) e solte-o no nó de `gridLayout` na janela **Outline**. Altere a propriedade **Id** de `EditText` para `amountEditText`. O componente `EditText` é colocado na *segunda* coluna da *primeira* linha de `GridLayout`.
3. Arraste outro componente `TextView` **Medium** para o nó de `gridLayout` na janela **Outline** e altere a propriedade **Id** para `amountDisplayTextView`. O novo componente `TextView` é inicialmente colocado na *primeira* coluna da *segunda* linha de `GridLayout`. Para colocá-lo na *segunda* coluna da *primeira* linha de `GridLayout`, configure as propriedades **Row** e **Column** desse componente `TextView` (localizado na seção **Layout Parameters** da janela **Properties**) com os valores 0 e 1, respectivamente.

Passo 2: Adicionando visualizações à segunda linha

Em seguida, você vai adicionar componentes `TextView` e `SeekBar` a `GridLayout`. Para isso:

1. Arraste um componente `TextView` (`customPercentTextView`) **Medium** da seção **Form Widgets** da **Palette** para o nó de `gridLayout` na janela **Outline**.
2. Arraste um componente `SeekBar` (`customTipSeekBar`) da seção **Form Widgets** da **Palette** para o nó de `gridLayout` na janela **Outline**.

Passo 3: Adicionando visualizações à terceira linha

Em seguida, você vai adicionar a `GridLayout` um componente `LinearLayout` contendo dois componentes `TextView`. Para isso:

1. Da seção **Layouts** da **Palette**, arraste um componente **Linear Layout (Horizontal)** (`percentLinearLayout`) para o nó de `gridLayout` na janela **Outline**.
2. Arraste um componente **TextView** (`percent15TextView`) **Medium** para o nó de `percentLinearLayout` na janela **Outline**. Isso aninha o novo componente **TextView** no componente **LinearLayout**.
3. Arraste outro componente **TextView** (`percentCustomTextView`) **Medium** para o nó de `percentLinearLayout` na janela **Outline**.
4. O componente `percentLinearLayout` e seus dois componentes **TextView** aninhados devem ser colocados na segunda coluna do **GridLayout**. Para fazer isso, selecione o componente `percentLinearLayout` na janela **Outline** e configure sua propriedade **Column** como 1.

Passo 4: Adicionando visualizações à quarta linha

Em seguida, você vai adicionar a **GridLayout** um **TextView** e um **LinearLayout** contendo mais dois componentes **TextView**. Para isso:

1. Arraste um componente **TextView** (`tipTextView`) **Medium** para o nó de `gridLayout`.
2. Arraste um componente **Linear Layout (Horizontal)** (`tipLinearLayout`) para o nó de `gridLayout`.
3. Arraste dois componentes **TextView** (`tip15TextView` e `tipCustomTextView`) **Medium** para o nó de `tipLinearLayout`.

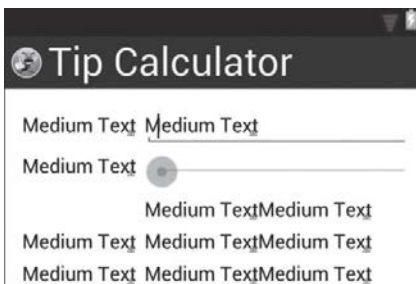
Passo 5: Adicionando visualizações à quinta linha

Para criar a última linha da interface gráfica do usuário, repita o passo 4 usando as propriedades **Id** `totalTextView`, `totalLinearLayout`, `total15TextView` e `totalCustomTextView`.

Exame do layout até o momento

A interface gráfica do usuário e a janela **Outline** devem agora aparecer como mostrado na Figura 3.6. Os símbolos de alerta mostrados no editor **Graphical Layout** e na janela **Outline** desaparecerão quando você concluir o projeto da interface, na Seção 3.4.5.

a) Projeto da interface gráfica do usuário até o momento



b) Janela Outline mostrando componentes do aplicativo Tip Calculator

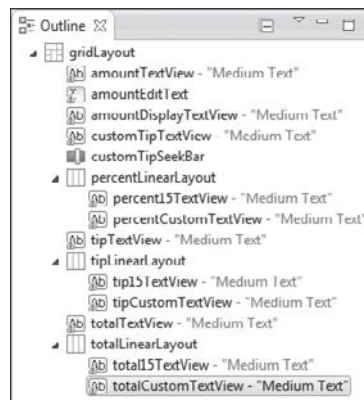


Figura 3.6 A interface gráfica do usuário e a janela **Outline** do IDE após a adição de todas as visualizações ao componente **GridLayout**.

3.4.5 Personalização das visualizações para concluir o projeto

Agora você vai concluir o projeto do aplicativo personalizando propriedades das views e criando várias strings e recursos de dimensionamento. Conforme você aprendeu na Seção 2.5, os valores de strings literais devem ser colocados no arquivo de recurso `strings.xml`. Da mesma forma, os valores numéricos literais que especificam dimensões das views (por exemplo, larguras, alturas e espaçamento) devem ser colocados no arquivo de recurso `dimens.xml`.

Passo 6: Especificando texto literal

Especifique o texto literal dos componentes `amountTextView`, `customPercentTextView`, `percent15TextView`, `percentCustomTextView`, `tipTextView` e `totalTextView`:

1. Selecione o componente `amountTextView` na janela **Outline**.
2. Na janela **Properties**, clique no botão de reticências ao lado da propriedade **Text**.
3. Na caixa de diálogo **Resource Chooser**, clique em **New String....**
4. Na caixa de diálogo **Create New Android String**, especifique **Amount** no campo **String** e **amount** no campo **New R.string**; em seguida, clique em **OK**.
5. Na caixa de diálogo **Resource Chooser**, clique em **OK** para configurar a propriedade **Text** de `amountTextView` com o recurso de string identificado como **amount**.

Repita as tarefas anteriores para os outros componentes `TextView` usando os valores mostrados na Fig. 3.7.

View	String	Nova R.string
<code>customPercentTextView</code>	Custom %	<code>custom_tip_percentage</code>
<code>percent15TextView</code>	15%	<code>fifteen_percent</code>
<code>percentCustomTextView</code>	18%	<code>eighteen_percent</code>
<code>tipTextView</code>	Tip	<code>tip</code>
<code>totalTextView</code>	Total	<code>total</code>

Figura 3.7 Valores de recurso de string e IDs de recurso.

Passo 7: Alinhando os componentes `TextView` à direita na coluna da esquerda

Na Fig. 3.3, cada um dos componentes `TextView` da coluna da esquerda é alinhado à direita. Para `amountTextView`, `customPercentTextView`, `tipTextView` e `totalTextView`, configure a propriedade **Gravity** do layout como **right** – localizada na seção **Layout Parameters** na janela **Properties**.

Passo 8: Configurando a propriedade **Label For de `amountTextView`**

Geralmente, cada componente `EditText` deve ter um componente `TextView` descritivo que ajude o usuário a entender a finalidade do componente `EditText` (também é útil para acessibilidade) – caso contrário, a *Android Lint* emitirá um alerta. Para corrigir isso, configure a propriedade **Label For** do componente `TextView` com o **Id** do componente `EditText` associado. Selecione o componente `amountTextView` e configure sua propriedade de **Label For** (na seção **View** da janela **Properties**) como

```
@+id/amountEditText
```


O sinal + é obrigatório porque o componente `TextView` é definido *antes* do componente `EditText` na interface gráfica do usuário; portanto, o componente `EditText` ainda não existe quando o Android converte o código XML do layout na interface.

Passo 9: Configurando o componente `amountEditText`

No aplicativo final, o componente `amountEditText` fica *oculto* atrás do componente `amountDisplayTextView` e é configurado de modo a permitir a inserção somente de *dígitos* pelo usuário. Selecione o componente `amountEditText` e configure as seguintes propriedades:

1. Na seção **Layout Parameters** da janela **Properties**, configure **Width** e **Height** como `wrap_content`. Isso indica que o componente `EditText` deve ser grande o suficiente apenas para caber seu conteúdo, incluindo qualquer preenchimento.
2. Remova o valor de **Gravity** do layout `fill_horizontal`, deixando o valor da propriedade em branco. Vamos discutir `fill_horizontal` no próximo passo.
3. Remova o valor da propriedade **Ems**, a qual indica a largura do componente `EditText`, medida em caracteres M maiúsculos da fonte da view. Em nosso componente `GridLayout`, isso faz a segunda coluna ser muito estreita; portanto, removemos essa configuração padrão.
4. Na seção **TextView** da janela **Properties**, configure **Digits** como `0123456789` – isso permite que *apenas* dígitos sejam inseridos, mesmo que o teclado numérico contenha botões de subtração (-), vírgula (,), ponto final (.) e espaço. Por padrão, a propriedade **Digits** *não* é exibida na janela **Properties**, pois é considerada uma propriedade avançada. Para exibi-la, clique no botão de alternância **Show Advanced Properties** (🔍) na parte superior da janela **Properties**.
5. Restringimos o valor da conta a um máximo de *seis* dígitos – portanto, o maior valor de conta aceito é `9999.99`. Na seção **TextView** da janela **Properties**, configure a propriedade **Max Length** como `6`.

Passo 10: Configurando o componente `amountDisplayTextView`

Para concluir a formatação do componente `amountDisplayTextView`, selecione-o e configure as seguintes propriedades:

1. Na seção **Layout Parameters** da janela **Properties**, configure **Width** e **Height** como `wrap_content` para indicar que o componente `TextView` deve ser grande o suficiente para caber seu conteúdo.
2. Remova o valor da propriedade **Text** – vamos exibir texto via programação aqui.
3. Na seção **Layout Parameters** da janela **Properties**, configure **Gravity** do layout como `fill_horizontal`. Isso indica que o componente `TextView` deve ocupar todo o espaço horizontal restante nessa linha do `GridLayout`.
4. Na seção **View**, configure **Background** como `@android:color/holo_blue_bright`. Essa é uma das várias *cores predefinidas* (cada uma começa com `@android:color`) no tema *Holo* do Android. Quando você começa a digitar o valor da propriedade **Background**, aparece uma lista suspensa das cores disponíveis do tema. Você também pode usar qualquer *cor personalizada*, criada a partir de uma combinação dos componentes vermelho, verde e azul, chamados de **valores RGB** – cada um é um valor inteiro no intervalo de 0 a 255 que define a quantidade de vermelho, verde e azul na cor, respectivamente. As cores personalizadas são

definidas no *formato hexadecimal (base 16)*; portanto, os componentes RGB são valores no intervalo de 00 a FF. O Android também aceita valores *alfa (transparência)* no intervalo de 0 (*completamente transparente*) a 255 (*completamente opaco*). Para usar valores alfa, você especifica a cor no formato #AARRGGBB, em que os dois primeiros dígitos hexadecimais representam o valor alfa. Se os dois dígitos de cada componente de cor forem iguais, você pode usar os formatos abreviados #RGB ou #ARGB. Por exemplo, #9AC é tratado como #99AACC e #F9AC é tratado como #FF99AACC.

5. Por fim, você vai acrescentar algum preenchimento em torno do componente TextView. Para isso, você vai criar um novo *recurso de dimensão* chamado `textview_padding`, o qual você vai usar várias vezes na interface gráfica do usuário. A propriedade **Padding** de uma visualização especifica um espaço em todos os lados de seu conteúdo. Na seção **View** da janela **Properties**, clique no botão de reticências da propriedade **Padding**. Clique em **New Dimension...** para criar um novo *recurso de dimensão*. Especifique `textview_padding` para **Name**, `8dp` para **Value** e clique em **OK**; então, selecione seu novo *recurso de dimensão* e clique em **OK**.

Passo 11: Configurando o componente `customPercentTextView`

Observe que o componente `customPercentTextView` está alinhado com a parte superior do cursor de `customTipSeekBar`. Isso parecerá melhor se for *centralizado verticalmente*. Para tanto, na seção **Layout Parameters** da janela **Properties**, modifique o valor de **Gravity** para

```
right|center_vertical
```

O caractere de *barra vertical* (|) é usado para separar *vários* valores de **Gravity** – neste caso, indicando que o componente TextView deve ser *alinhado à direita e centralizado verticalmente* dentro da célula da grade. Além disso, configure as propriedades **Width** e **Height** de `customPercentTextView` como `wrap_content`.

Passo 12: Configurando o componente `customTipSeekBar`

Por padrão, o intervalo de um componente `SeekBar` é de 0 a 100, e seu valor atual é indicado por sua propriedade **Progress**. Este aplicativo permite porcentagens de gorjeta personalizadas de 0 a 30 e especifica 18 como padrão. Configure a propriedade **Max** do componente como 30 e a propriedade **Progress** como 18. Além disso, configure as propriedades **Width** e **Height** como `wrap_content`.

Passo 13: Configurando os componentes `percent15TextView` e `percentCustomTextView`

Lembre-se de que o componente `GridLayout` *não* permite especificar como um componente view deve ser dimensionado em relação aos demais em determinada linha. Foi por isso que colocamos os componentes `percent15TextView` e `percentCustomTextView` em um `LinearLayout`, o qual *permite dimensionamento proporcional*. A propriedade **Weight** do layout de uma visualização (em certos layouts, como o `LinearLayout`) especifica a importância relativa daquela view com respeito às outras views do layout. Por padrão, todas as visualizações têm a propriedade **Weight** definida como 0.

Neste layout, configuramos **Weight** como 1 para `percent15TextView` e `percentCustomTextView` – isso indica que eles têm a mesma importância; portanto, devem ser dimensionados igualmente. Por padrão, quando adicionamos o componente

percentLinearLayout ao GridLayout, a propriedade **Gravity** de seu layout foi configurada como `fill_horizontal`; portanto, o layout ocupa o espaço restante na terceira linha. Quando o componente `LinearLayout` é alongado para preencher o restante da linha, os componentes `TextView` ocupam cada um *metade* da largura do `LinearLayout`.

Quisemos também que cada componente `TextView` centralizasse seu texto. Para isso, na seção **TextView** da janela **Properties**, configuramos a propriedade **Gravity** como `center`. Isso especifica o alinhamento de texto do componente `TextView`, enquanto a propriedade **Gravity** do *layout* especifica como um componente *view* é alinhado com relação ao *layout*.

Passo 14: Configurando os componentes `tip15TextView`, `tipCustomTextView`, `total15TextView` e `totalCustomTextView`

Para finalizar esses quatro componentes `TextView`, execute as tarefas a seguir em cada um deles:

1. Selecione o componente `TextView`.
2. Exclua seu valor de **Text** – vamos configurar isso via programação.
3. Configure a propriedade **Background** como `@android:color/holo_orange_light`.
4. Configure a propriedade **Gravity** do layout como `center`.
5. Configure a propriedade **Weight** do layout como 1.
6. Configure a propriedade **Width** do layout como 0dp – isso permite que o layout use a propriedade **Weight** para determinar a largura da *view*.
7. Configure a propriedade **Gravity** do componente `TextView` como `center`.
8. Configure a propriedade **Padding** do componente `TextView` como `@dimen/textview_padding` (o *recurso de dimensão* que você criou em um passo anterior).

Observe que *não* há *espaço horizontal* entre os componentes `TextView` em `tiplLinearLayout` e `totalLinearLayout`. Para corrigir isso, você vai especificar uma margem direita de 8dp para `tip15TextView` e para `total15TextView`. Na seção **Layout Parameters** da janela **Properties**, expanda a seção **Margin** e, então, configure a margem **Right** (direita) como 8dp, criando um novo *recurso de dimensão* chamado `textview_margin`. Em seguida, use esse recurso para configurar a margem **Right** de `total15TextView`.

Passo 15: Centralizando os componentes `tipTextView` e `totalTextView` verticalmente

Para centralizar os componentes `tipTextView` e `totalTextView` verticalmente com as outras visualizações em suas respectivas linhas, modifique as propriedades **Gravity** de seus layouts de `right` para

```
right|center_vertical
```

Quando você faz isso para o componente `totalTextView`, o `GridLayout` centraliza esse componente verticalmente no *espaço restante da quinta linha até a parte inferior da tela*. Para corrigir esse problema, arraste uma **View Space** (na seção **Layout** da **Palette**) para o nó de `GridLayout` na janela **Outline**. Isso cria uma sexta linha que ocupa o restante da tela. Conforme seu nome implica, um componente de visualização **Space** (espaço) ocupa espaço em uma interface gráfica do usuário. Agora a interface gráfica do usuário deve aparecer como na Figura 3.8.

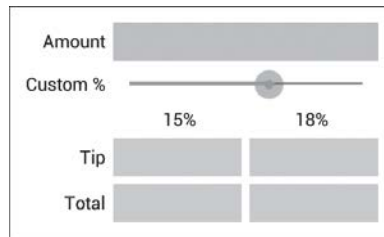


Figura 3.8 Projeto final da interface gráfica do usuário.

3.5 Adição de funcionalidade ao aplicativo

A classe `MainActivity` (Figs. 3.9 a 3.16) implementa as funcionalidades do aplicativo `Tip Calculator`. Ela calcula os valores das gorjetas de 15% e os de porcentagem personalizada e o total da conta, e os exibe no formato de moeda corrente específico da localidade. Para ver o arquivo, abra `src/com.deitel.tipcalculator` e clique duas vezes em `MainActivity.java`. Você precisará digitar a maior parte do código das Figs. 3.9 a 3.16.

As instruções `package` e `import`

A Figura 3.9 mostra a instrução `package` e as instruções `import` em `MainActivity.java`. A instrução `package` na linha 3 foi inserida quando você criou o projeto. Quando um arquivo Java é aberto no IDE, as instruções `import` estão recolhidas – uma aparece com  à sua esquerda. Clique em  para ver a lista completa de instruções `import`.

```

1 // MainActivity.java
2 // Calcula valores de conta usando porcentagens de 15% e personalizadas.
3 package com.deitel.tipcalculator;
4
5 import java.text.NumberFormat; // para formatação de moeda corrente
6
7 import android.app.Activity; // classe base para atividades
8 import android.os.Bundle; // para salvar informações de estado
9 import android.text.Editable; // para tratamento de eventos de EditText
10 import android.text.TextWatcher; // receptor de EditText
11 import android.widget.EditText; // para entrada do valor da conta
12 import android.widget.SeekBar; // para alterar a porcentagem de gorjeta personalizada
13 import android.widget.SeekBar.OnSeekBarChangeListener; // receptor de SeekBar
14 import android.widget.TextView; // para exibir texto
15
```

Figura 3.9 Instruções `package` e `import` de `MainActivity`.

As linhas 5 a 14 importam as classes e interfaces utilizadas pelo aplicativo:

- A classe `NumberFormat` do pacote `java.text` (linha 5) fornece recursos de formatação numérica, como formatos de moeda corrente *específicos da localidade* e formatos de porcentagem.
- A classe `Activity` do pacote `android.app` (linha 7) fornece os *métodos de ciclo de vida* básicos de um aplicativo – vamos discuti-los em breve.
- A classe `Bundle` do pacote `android.os` (linha 8) representa a *informação de estado* de um aplicativo. O Android oferece a um aplicativo a oportunidade de *salvar seu*

estado antes que outro aplicativo apareça na tela. Isso pode ocorrer, por exemplo, quando o usuário *ativa outro aplicativo* ou *recebe uma ligação telefônica*. O aplicativo que está atualmente na tela em dado momento está no *primeiro plano* (o usuário pode interagir com ele e o aplicativo consome a CPU) e todos os outros estão no *segundo plano* (o usuário não pode interagir com eles e, normalmente, eles não estão consumindo a CPU). Quando outro aplicativo vem para o primeiro plano, o que estava nele anteriormente tem a oportunidade de *salvar seu estado* ao ser enviado para o segundo plano.

- A interface `Editable` do pacote `android.text` (linha 9) permite modificar o conteúdo e a marcação de texto em uma interface gráfica do usuário.
- Você implementa a interface `TextWatcher` do pacote `android.text` (linha 10) para responder a eventos quando o usuário altera o texto em um componente `EditText`.
- O pacote `android.widget` (linhas 11 a 14) contém os *widgets* (isto é, componentes de visualização) e layouts utilizados nas interfaces gráficas do usuário do Android. Este aplicativo usa os widgets `EditText` (linha 11), `SeekBar` (linha 12) e `TextView` (linha 14).
- Você implementa a interface `SeekBar.OnSeekBarChangeListener` do pacote `android.widget` (linha 13) para responder ao movimento, feito pelo usuário, do *cursor* de um elemento `SeekBar`.

Ao escrever código com várias classes e interfaces, é possível usar o comando **Source > Organize Imports** do IDE para permitir que o IDE insira instruções `import` para você. Para casos nos quais o mesmo nome de classe ou interface aparece em mais de um pacote, o IDE permitirá selecionar a instrução `import` apropriada.

Activity do aplicativo Tip Calculator e o ciclo de vida da atividade

A classe `MainActivity` (Figs. 3.10 a 3.16) é a subclasse de `Activity` do aplicativo `Tip Calculator`. Quando você criou o projeto `Tip Calculator`, o IDE gerou essa classe como uma subclasse de `Activity` e sobrescreveu o método `onCreate` herdado da classe `Activity` (Fig. 3.11). Toda subclasse de `Activity` *deve* sobrescrever esse método. O código padrão da classe `MainActivity` também incluiu um método `onCreateOptionsMenu`, o qual removemos porque não é usado neste aplicativo. Vamos discutir `onCreate` em breve.

```
16 // classe MainActivity do aplicativo Tip Calculator
17 public class MainActivity extends Activity
18 {
```

Figura 3.10 A classe `MainActivity` é uma subclasse de `Activity`.

Variáveis de classe e variáveis de instância

As linhas 20 a 32 da Fig. 3.11 declaram as variáveis da classe `MainActivity`. Os objetos `NumberFormat` (linhas 20 a 23) são usados para formatar valores de moeda corrente e porcentagens, respectivamente. O método estático `getCurrencyInstance` de `NumberFormat` retorna um objeto `NumberFormat` que formata valores como moeda corrente usando a *localidade padrão* do dispositivo. Do mesmo modo, o método estático `getPercentInstance` formata valores como porcentagens usando a *localidade padrão* do dispositivo.

```

19 // formatadores de moeda corrente e porcentagem
20 private static final NumberFormat currencyFormat =
21     NumberFormat.getCurrencyInstance();
22 private static final NumberFormat percentFormat =
23     NumberFormat.getPercentInstance();
24
25 private double billAmount = 0.0; // valor da conta inserido pelo usuário
26 private double customPercent = 0.18; // porcentagem de gorjeta padronizada inicial
27 private TextView amountDisplayTextView; // mostra o valor da conta formatado
28 private TextView percentCustomTextView; // mostra a porcentagem de gorjeta personalizada
29 private TextView tip15TextView; // mostra gorjeta de 15%
30 private TextView total15TextView; // mostra o total com 15% de gorjeta
31 private TextView tipCustomTextView; // mostra o valor da gorjeta personalizada
32 private TextView totalCustomTextView; // mostra o total com a gorjeta personalizada
33

```

Figura 3.11 Variáveis de instância da classe MainActivity.

O valor da conta inserido pelo usuário no componente `amountEditText` será lido e armazenado como um valor `double` em `billAmount` (linha 25). A porcentagem de gorjeta personalizada (um valor inteiro no intervalo de 0 a 30), que o usuário define movendo o *cursor* do componente `SeekBar`, será multiplicado por 0.01 para criar um valor `double` para uso nos cálculos e, então, armazenado em `customPercent` (linha 26). Por exemplo, se você selecionar 25 com o componente `SeekBar`, `customPercent` armazenará 0.25, de modo que o aplicativo multiplicará o valor da conta por 0.25 para calcular a gorjeta de 25%.

A linha 27 declara o componente `TextView` que exibe o valor da conta formatado em moeda corrente. A linha 28 declara o componente `TextView` que exibe a porcentagem de gorjeta personalizada com base na posição do *cursor* do elemento `SeekBar` (observe o valor de 18% na Fig. 3.1(a)). As variáveis nas linhas 29 a 32 vão fazer referência aos componentes `TextView` nos quais o aplicativo exibe as gorjetas e os totais calculados.

Sobrescrevendo o método `onCreate` da classe `Activity`

O método `onCreate` (Fig. 3.12) – gerado automaticamente com as linhas 38 e 39 quando você cria o projeto do aplicativo – é chamado pelo sistema quando uma atividade é iniciada. O método `onCreate` normalmente inicializa as variáveis de instância e os componentes de visualização de `Activity`. Esse método deve ser o mais simples possível para que o aplicativo *seja carregado rapidamente*. Na verdade, se o aplicativo demorar mais de *cinco segundos* para carregar, o sistema operacional vai exibir uma **caixa de diálogo ANR (Application Not Responding)** – dando ao usuário a opção de *terminar o aplicativo à força*. Você vai aprender a evitar esse problema no Capítulo 8.

```

34 // chamado quando a atividade é criada
35 @Override
36 protected void onCreate(Bundle savedInstanceState)
37 {
38     super.onCreate(savedInstanceState); // chama a versão da superclasse
39     setContentView(R.layout.activity_main); // infla a interface gráfica do usuário
40
41     // obtém referências para os componentes TextView
42     // com que MainActivity interage via programação
43     amountDisplayTextView =
44         (TextView) findViewById(R.id.amountDisplayTextView);
45     percentCustomTextView =
46         (TextView) findViewById(R.id.percentCustomTextView);

```

Figura 3.12 Sobrescrevendo o método `onCreate` de `Activity`.

```

47 tip15TextView = (TextView) findViewById(R.id.tip15TextView);
48 total15TextView = (TextView) findViewById(R.id.total15TextView);
49 tipCustomTextView = (TextView) findViewById(R.id.tipCustomTextView);
50 totalCustomTextView =
51     (TextView) findViewById(R.id.totalCustomTextView);
52
53 // atualiza a interface gráfica do usuário com base em billAmount e customPercent
54 amountDisplayTextView.setText(
55     currencyFormat.format(billAmount));
56 updateStandard(); // atualiza os componentes TextView de gorjeta de 15%
57 updateCustom(); // atualiza os componentes TextView de gorjeta personalizada
58
59 // configura TextWatcher de amountEditText
60 EditText amountEditText =
61     (EditText) findViewById(R.id.amountEditText);
62 amountEditText.addTextChangedListener(amountEditTextWatcher);
63
64 // configura OnSeekBarChangeListener de customTipSeekBar
65 SeekBar customTipSeekBar =
66     (SeekBar) findViewById(R.id.customTipSeekBar);
67 customTipSeekBar.setOnSeekBarChangeListener(customSeekBarListener);
68 } // fim do método onCreate
69

```

Figura 3.12 Sobrescrevendo o método onCreate de Activity.

Parâmetro Bundle de onCreate

Durante a execução do aplicativo, o usuário pode alterar a configuração do dispositivo *girando-o* ou *abrindo um teclado físico*. Para proporcionar uma boa experiência, o aplicativo deve continuar a funcionar naturalmente, apesar das mudanças na configuração. Quando o sistema chama onCreate, ele passa um argumento **Bundle** contendo o estado salvo da atividade, se houver um. Normalmente, você salva estado nos métodos onPause ou onSaveInstanceState de Activity (demonstrado em aplicativos posteriores). A linha 38 chama o método onCreate da superclasse, o que é *obrigatório* ao sobrescrever onCreate.

A classe R gerada contém IDs de recurso

Quando você constrói a interface gráfica do usuário de seu aplicativo e adiciona *recursos* (como strings no arquivo strings.xml ou visualizações no arquivo activity_main.xml) no aplicativo, o IDE gera uma classe chamada **R** que contém *classes aninhadas* representando cada tipo de recurso presente na pasta res de seu projeto. Você pode encontrar essa classe na **pasta gen** de seu projeto, a qual contém arquivos de código-fonte gerados. As classes aninhadas são declaradas com a instrução static, de modo que você pode acessá-las em seu código com *R.NomeDaClasse*. Dentro das classes aninhadas da classe R, o IDE cria constantes static final int que permitem fazer referência aos recursos de seu aplicativo via programação, a partir de seu código (conforme vamos discutir daqui a pouco). Algumas das classes aninhadas na classe R incluem:

- classe **drawable** – contém constantes para todos os itens drawable, como *imagens*, que você coloca nas várias pastas drawable da pasta res de seu aplicativo
- classe **id** – contém constantes para os *componentes view* em seus arquivos de *layout XML*
- classe **layout** – contém constantes que representam cada *arquivo de layout* em seu projeto (como activity_main.xml)
- classe **string** – contém constantes para cada String no arquivo strings.xml.

Inflando a interface gráfica do usuário

A chamada de `setContentView` (linha 39) recebe a constante `R.layout.activity_main` para indicar qual arquivo XML representa a interface gráfica de `MainActivity` – neste caso, a constante representa o arquivo `main.xml`. O método `setContentView` usa essa constante para carregar o documento XML correspondente, o qual é então analisado via parsing e convertido na interface gráfica do usuário do aplicativo. Esse processo é conhecido como **inflar** (ou expandir) a interface gráfica do usuário.

Obtendo referências para os widgets

Uma vez *inflado* o layout, você pode *obter referências para os widgets individuais* a fim de poder interagir com eles via programação. Para isso, use o método `findViewById` da classe `Activity`. Esse método recebe uma constante `int` representando o `id` de uma visualização específica e retorna uma referência para esta. O nome da constante `R.id` de cada visualização é determinado pela propriedade `id` do componente, que você especificou ao projetar a interface do usuário. Por exemplo, a constante de `amountEditText` é `R.id.amountEditText`.

As linhas 43 a 51 obtêm referências para os componentes `TextView` que são alterados pelo aplicativo. As linhas 43 e 44 obtêm uma referência para o elemento `amountDisplayTextView` que é atualizado quando o usuário insere o valor da conta. As linhas 45 e 46 obtêm uma referência para o componente `percentCustomTextView` que vai ser atualizado quando o usuário alterar a porcentagem de gorjeta personalizada. As linhas 47 a 51 obtêm referências para os componentes `TextView` em que são exibidas as gorjetas e os totais calculados.

Exibindo valores iniciais nos componentes TextView

As linhas 54 e 55 definem o texto de `amountDisplayTextView` como o `billAmount` inicial (0.00) em um formato de moeda corrente *específico da localidade*, chamando o **método `format`** do objeto `currencyFormat`. Em seguida, as linhas 56 e 57 chamam os métodos `updateStandard` (Fig. 3.13) e `updateCustom` (Fig. 3.14) para exibir valores iniciais nos componentes `TextView` de gorjeta e total.

Registrando os objetos receptores de eventos

As linhas 60 e 61 obtêm uma referência para `amountEditText`, e a linha 62 chama seu método `addTextChangedListener` para registrar o objeto `TextChangedListener` que vai responder aos *eventos* gerados quando o *usuário altera o texto* no componente `EditText`. Definimos esse receptor (Fig. 3.16) como um *objeto de classe interna anônima* atribuído à variável de instância `amountEditTextWatcher`.*

As linhas 65 e 66 obtêm uma referência para o componente `customTipSeekBar`, e a linha 67 chama seu método `setOnSeekBarChangeListener` para registrar o objeto `OnSeekBarChangeListener` que vai responder aos *eventos* gerados quando o usuário mover o *cursor* de `customTipSeekBar` para mudar a porcentagem de gorjeta personalizada. Definimos esse receptor (Fig. 3.15) como um *objeto de classe interna anônima* atribuído à variável de instância `customSeekBarListener`.

Método `updateStandard` da classe `MainActivity`

O método `updateStandard` (Figura 3.13) atualiza os componentes `TextView` de gorjeta de 15% e do total sempre que o usuário *altera* o total da conta. O método usa o valor de `billAmount` para calcular a gorjeta e o total da conta com a gorjeta. As linhas 78 e 79 exibem os valores no formato de moeda corrente.

* N. de T. O termo “receptor de eventos” refere-se a “event listener” na documentação original.


```

70 // atualiza os componentes TextView de gorjeta de 15%
71 private void updateStandard()
72 {
73     // calcula a gorjeta de 15% e o total
74     double fifteenPercentTip = billAmount * 0.15;
75     double fifteenPercentTotal = billAmount + fifteenPercentTip;
76
77     // exibe a gorjeta de 15% e o total formatados como moeda corrente
78     tip15TextView.setText(currencyFormat.format(fifteenPercentTip));
79     total15TextView.setText(currencyFormat.format(fifteenPercentTotal));
80 } // fim do método updateStandard
81

```

Figura 3.13 O método `updateStandard` calcula e exibe a gorjeta de 15% e o total.

Método `updateCustom` da classe `MainActivity`

O método `updateCustom` (Fig. 3.14) atualiza os componentes `TextView` de gorjeta personalizada e total com base na porcentagem da gorjeta selecionada pelo usuário com o elemento `customTipSeekBar`. A linha 86 define o texto de `percentCustomTextView` com o valor de `customPercent` formatado como porcentagem. As linhas 89 e 90 calculam `customTip` e `customTotal`. Então, as linhas 93 e 94 exibem os valores no formato de moeda corrente.

```

82 // atualiza os componentes TextView de gorjeta personalizada e total
83 private void updateCustom()
84 {
85     // mostra customPercent em percentCustomTextView, formatado como %
86     percentCustomTextView.setText(percentFormat.format(customPercent));
87
88     // calcula a gorjeta personalizada e o total
89     double customTip = billAmount * customPercent;
90     double customTotal = billAmount + customTip;
91
92     // exibe a gorjeta personalizada e o total, formatados como moeda corrente
93     tipCustomTextView.setText(currencyFormat.format(customTip));
94     totalCustomTextView.setText(currencyFormat.format(customTotal));
95 } // fim do método updateCustom
96

```

Figura 3.14 O método `updateCustom` calcula e exibe a gorjeta personalizada e o total.

Classe interna anônima que implementa a interface `OnSeekBarChangeListener`

As linhas 98 a 120 da Figura 3.15 criam o objeto de *classe interna anônima* chamado `customSeekBarListener` que responde aos *eventos* de `customTipSeekBar`. Se não estiver familiarizado com *classes internas anônimas*, visite a seguinte página:

<http://bit.ly/AnonymousInnerClasses>

A linha 67 (Fig. 3.12) registrou `customSeekBarListener` como objeto de *tratamento de eventos* `OnSeekBarChangeListener` de `customTipSeekBar`. Por clareza, definimos todos os objetos de tratamento de eventos (menos os mais simples) dessa maneira, para não congestionar o método `onCreate` com esse código.

```

97 // chamado quando o usuário muda a posição de SeekBar
98 private OnSeekBarChangeListener customSeekBarListener =
99     new OnSeekBarChangeListener()

```

Figura 3.15 Classe interna anônima que implementa a interface `OnSeekBarChangeListener` para responder aos eventos do componente `customSeekBar`. (continua)


```

100     {
101         // atualiza customPercent e chama updateCustom
102         @Override
103         public void onProgressChanged(SeekBar seekBar, int progress,
104             boolean fromUser)
105         {
106             // configura customPercent com a posição do cursor de SeekBar
107             customPercent = progress / 100.0;
108             updateCustom(); // atualiza os componentes TextView de gorjeta personalizada
109         } // fim do método onProgressChanged
110
111         @Override
112         public void onStartTrackingTouch(SeekBar seekBar)
113         {
114             } // fim do método onStartTrackingTouch
115
116         @Override
117         public void onStopTrackingTouch(SeekBar seekBar)
118         {
119             } // fim do método onStopTrackingTouch
120     }; // fim de OnSeekBarChangeListener
121

```

Figura 3.15 Classe interna anônima que implementa a interface `OnSeekBarChangeListener` para responder aos eventos do componente `customSeekBar`.

Sobrescrevendo o método `onProgressChanged` da interface `OnSeekBarChangeListener`

As linhas 102 a 119 implementam os métodos da interface `OnSeekBarChangeListener`. O método `onProgressChanged` é chamado sempre que a posição do *cursor* do componente `SeekBar` *muda*. A linha 107 calcula `customPercent` usando o parâmetro `progress` do método – um valor `int` representando a posição do *cursor* do componente `SeekBar`. Dividimos isso por 100,0 para obter a porcentagem personalizada. A linha 108 chama o método `updateCustom` para recalcular e exibir a gorjeta personalizada e o total.

Sobrescrevendo os métodos `onStartTrackingTouch` e `onStopTrackingTouch` da interface `OnSeekBarChangeListener`

A linguagem Java exige que *cada* método de uma *interface* que você *implemente* seja sobrescrito. Este aplicativo *não* precisa saber quando o usuário *começa* a mover o cursor do controle deslizante (`onStartTrackingTouch`) ou quando *para* de movê-lo (`onStopTrackingTouch`); portanto, fornecemos simplesmente um corpo *vazio* para cada um deles (linhas 111 a 119) para *cumprir o contrato da interface*.

Classe interna anônima que implementa a interface `TextWatcher`

As linhas 123 a 156 da Figura 3.16 criam o objeto de *classe interna anônima* `amountEditTextWatcher` que responde aos *eventos* de `amountEditText`. A linha 62 registrou esse objeto para *receber* os eventos de `amountEditText` que ocorrem quando o texto muda.

Sobrescrevendo o método `onTextChanged` da interface `TextWatcher`

O método `onTextChanged` (linhas 126 a 144) é chamado quando o texto do componente `amountEditText` é *modificado*. O método recebe quatro parâmetros. Neste exemplo, usamos apenas `CharSequence`s, que contém uma cópia do texto de `amountEditText`. Os outros parâmetros indicam que os `count` caracteres, a partir de `start`, *substituíram* o texto anterior de comprimento `before`.

```

122 // objeto de tratamento de eventos que responde aos eventos de amountEditText
123 private TextWatcher amountEditTextWatcher = new TextWatcher()
124 {
125     // chamado quando o usuário insere um número
126     @Override
127     public void onTextChanged(CharSequence s, int start,
128         int before, int count)
129     {
130         // converte o texto de amountEditText em um valor double
131         try
132         {
133             billAmount = Double.parseDouble(s.toString()) / 100.0;
134         } // fim do try
135         catch (NumberFormatException e)
136         {
137             billAmount = 0.0; // o padrão, caso ocorra uma exceção
138         } // fim do catch
139
140         // exibe o valor da conta formatado como moeda corrente
141         amountDisplayTextView.setText(currencyFormat.format(billAmount));
142         updateStandard(); // atualiza os componentes TextView de gorjeta de 15%
143         updateCustom(); // atualiza os componentes TextView de gorjeta personalizada
144     } // fim do método onTextChanged
145
146     @Override
147     public void afterTextChanged(Editable s)
148     {
149     } // fim do método afterTextChanged
150
151     @Override
152     public void beforeTextChanged(CharSequence s, int start, int count,
153         int after)
154     {
155     } // fim do método beforeTextChanged
156 }; // fim de amountEditTextWatcher
157 } // fim da classe MainActivity

```

Figura 3.16 Classe interna anônima que implementa a interface `TextWatcher` para responder aos eventos do componente `amountEditText`.

A linha 133 converte a entrada do usuário de `amountEditText` em um valor `double`. Permitimos ao usuário inserir somente números inteiros em centavos; portanto, dividimos o valor convertido por 100,0 para obtermos o valor da conta real – por exemplo, se o usuário digitar 2495, o valor da conta será 24.95. As linhas 142 e 143 chamam `updateStandard` e `updateCustom` para recalcular e exibir as gorjetas e os totais.

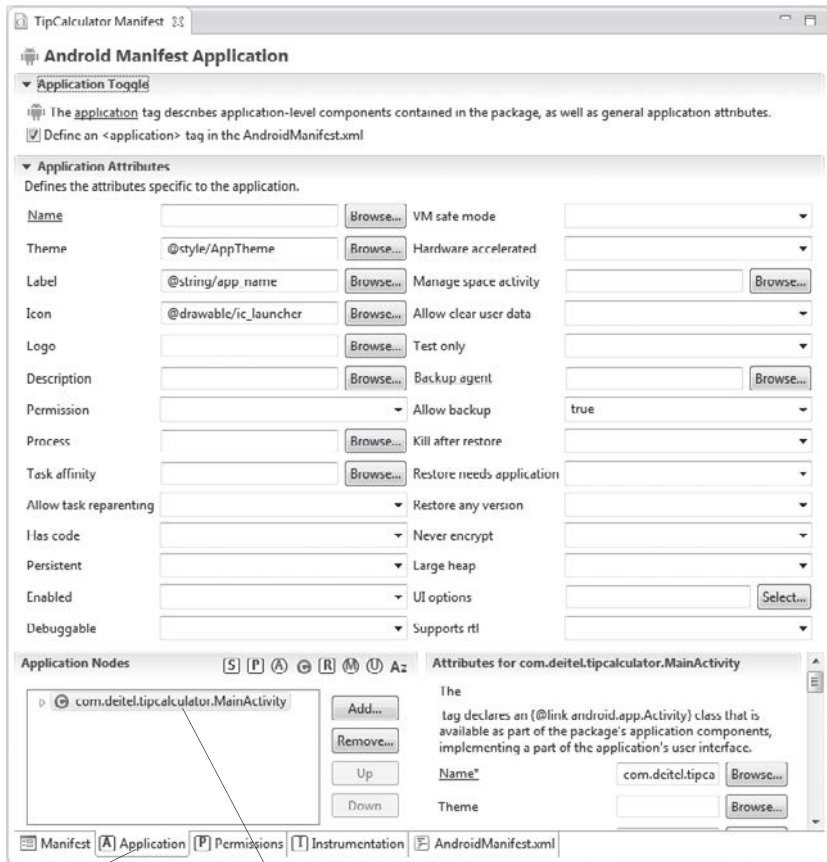
Outros métodos do componente `TextWatcher amountEditTextWatcher`

Este aplicativo *não* precisa saber que alterações estão para ser feitas no texto (`beforeTextChanged`) ou que o texto já foi alterado (`afterTextChanged`); portanto, simplesmente sobrescrevemos cada um desses métodos da interface `TextWatcher` com um corpo *vazio* (linhas 146 a 155) para *cumprir o contrato da interface*.

3.6 AndroidManifest.xml

Nesta seção, você vai modificar o arquivo `AndroidManifest.xml` para especificar que a atividade (Activity) deste aplicativo suporta apenas a orientação *retrato* de um dispositivo e que o *teclado numérico virtual sempre* deve permanecer na tela. Você vai usar o editor **Android Ma-**

nifest do IDE para especificar essas configurações. Para abrir o editor **Android Manifest**, clique duas vezes no arquivo `AndroidManifest.xml` do aplicativo no **Package Explorer**. Na parte inferior do editor, clique na guia **Application** (Fig. 3.17) e, então, selecione o nó `MainActivity` na seção **Application Nodes**, na parte inferior da janela. Isso exibe configurações para `MainActivity` na seção **Attributes for com.deitel.tipcalculator.MainActivity**.



Guia **Application** Selecione este nó para especificar configurações para `MainActivity` do aplicativo

Figura 3.17 Guia **Application** do editor **Android Manifest**.

Configurando `MainActivity` para a orientação retrato

Em geral, a maioria dos aplicativos deve suportar as orientações retrato e paisagem. Na orientação *retrato*, a altura do dispositivo é maior que sua largura. Na orientação *paisagem*, a largura é maior que a altura. No aplicativo *Tip Calculator*, girar o dispositivo para a orientação paisagem em um telefone típico faria o teclado numérico ocultar a maior parte da interface gráfica do usuário. Por isso, você vai configurar `MainActivity` para suportar apenas a orientação retrato. Na seção **Attributes for com.deitel.tipcalculator.MainActivity** do editor **Android Manifest**, role para baixo até a opção **Screen orientation** e selecione **portrait**.

Forçando o teclado numérico virtual a sempre aparecer para MainActivity

Quando o aplicativo Tip Calculator for executado, o teclado numérico deve ser exibido imediatamente e permanecer na tela o tempo todo. Na seção **Attributes for com.deitel.tipcalculator.MainActivity** do editor **Android Manifest**, role para baixo até a opção **Window soft input mode** e selecione **stateAlwaysVisible**. Observe que isso *não* exibirá o teclado numérico virtual se um teclado físico estiver presente.

3.7 Para finalizar

Neste capítulo, você criou seu primeiro aplicativo Android *interativo* – o Tip Calculator. Vimos um panorama dos recursos do aplicativo e, em seguida, você o testou para calcular gorjetas padrão e personalizadas com base no valor da conta digitado. Você seguiu passo a passo as instruções detalhadas para construir a interface gráfica do usuário do aplicativo usando o editor **Graphical Layout** do IDE Android Developer Tools, a janela **Outline** e a janela **Properties**. Percorremos também o código da subclasse MainActivity de Activity, a qual definiu as funcionalidades do aplicativo.

Na interface gráfica do aplicativo, você usou um componente GridLayout para organizar as visualizações em linhas e colunas. Você exibiu texto em componentes TextView e recebeu entrada de um componente EditText e de um componente SeekBar.

A classe MainActivity exigiu muitos recursos de programação orientada a objetos com Java, incluindo classes, objetos, métodos, interfaces, classes internas anônimas e herança. Explicamos a ideia de inflar a interface gráfica do usuário a partir de seu arquivo XML em sua representação na tela. Você aprendeu sobre a classe Activity do Android e parte do ciclo de vida de Activity. Em particular, você sobrescreveu o método onCreate para inicializar o aplicativo quando ele é iniciado. No método onCreate, você usou o método findViewById de Activity para obter referências para cada um dos componentes de visualização com que o aplicativo interage via programação. Definiu uma classe interna anônima para implementar a interface TextWatcher, permitindo ao aplicativo calcular novas gorjetas e totais quando o usuário altera o texto no componente EditText. Você também definiu uma classe interna anônima para implementar a interface OnSeekBarChangeListener, permitindo ao aplicativo calcular uma nova gorjeta personalizada e um novo total quando o usuário muda a porcentagem de gorjeta personalizada movendo o cursor do componente SeekBar.

Por último, abriu o arquivo AndroidManifest.xml no editor **Android Manifest** do IDE para especificar que MainActivity devia suportar somente a orientação retrato e que sempre devia exibir o teclado numérico.

O uso do editor **Graphical Layout** do IDE, da janela **Outline**, da janela **Properties** e do editor **Android Manifest** permitiu construir este aplicativo sem manipular o código XML nos arquivos de recurso do projeto e no arquivo AndroidManifest.xml.

No próximo capítulo, apresentamos as coleções enquanto construímos o aplicativo **Twitter® Searches**. Muitos aplicativos móveis exibem listas de itens. Você vai fazer isso usando um componente ListActivity contendo um elemento ListView vinculado a um elemento ArrayList<String>. Vai também armazenar dados do aplicativo como preferências do usuário e vai aprender a ativar o navegador Web do dispositivo para exibir uma página web.

Exercícios de revisão

- 3.1** Preencha os espaços em branco em cada um dos seguintes enunciados:
- a) Um componente _____ – frequentemente chamado de caixa de texto ou campo de texto em outras tecnologias de interface gráfica do usuário – é uma subclasse de `TextView` que pode exibir texto e aceitar entrada de texto do usuário.
 - b) Use um componente _____ para organizar os componentes da interface gráfica do usuário em células em uma grade retangular.
 - c) Quando se trabalha com layouts mais complexos como os componentes `TableLayout`, é difícil ver a estrutura aninhada do layout e colocar elementos nos locais aninhados corretos usando o Visual Layout Editor. A janela _____ torna essas tarefas mais fáceis, pois mostra a estrutura aninhada da interface gráfica do usuário. Portanto, em um componente `GridLayout`, você pode selecionar a linha apropriada e adicionar nela um elemento da interface.
 - d) A classe _____ do pacote `android.os` representa a informação de estado de um aplicativo.
 - e) Você implementa a interface _____ do pacote `android.text` para responder a eventos quando o usuário interage com um componente de `EditText`.
 - f) Normalmente, uma _____ distinta é associada a cada tela de um aplicativo.
 - g) O método _____ é chamado pelo sistema quando uma atividade está começando – isto é, quando sua interface gráfica do usuário está para ser exibida para que o usuário possa interagir com a atividade.
 - h) Quando você constrói a interface gráfica do usuário de seu aplicativo e adiciona recursos (como strings no arquivo `strings.xml` ou visualizações no arquivo `activity_main.xml`) no aplicativo, o IDE gera uma classe chamada _____ que contém classes aninhadas representando cada tipo de recurso presente na pasta `res` de seu projeto.
 - i) Classe _____ (aninhada na classe `R`) – contém constantes para todos os itens `drawable`, como imagens, que você coloca nas várias pastas `drawable` da pasta `res` de seu aplicativo.
 - j) Classe _____ (aninhada na classe `R`) – contém constantes para cada `String` no arquivo `strings.xml`.
 - k) Uma vez inflado o layout, você pode obter referências para os widgets individuais usando o método _____ de `Activity`. Esse método recebe uma constante `int` para uma visualização específica (isto é, um componente da interface gráfica do usuário) e retorna uma referência para ele.
- 3.2** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- a) O método `onCreate` normalmente inicializa as variáveis de instância de `Activity` e componentes da interface gráfica do usuário. Esse método deve ser o mais simples possível para que o aplicativo seja carregado rapidamente. Na verdade, se o aplicativo demorar mais de cinco segundos para carregar, o sistema operacional vai exibir uma caixa de diálogo ANR (Application Not Responding) – dando ao usuário a opção de terminar o aplicativo à força.
 - b) A propriedade relativa `Weight` de cada componente determina como ele deve ser dimensionado em relação aos outros componentes.
 - c) Como todos os programas Java, os aplicativos Android têm um método `main`.
 - d) Uma atividade ativa (ou em execução) é visível na tela e “tem o foco” – isto é, está no segundo plano.
 - e) Uma atividade parada é visível na tela e é provável que seja encerrada pelo sistema quando sua memória for necessária.

Respostas dos exercícios de revisão

- 3.1** a) EditText. b) GridLayout. c) Outline. d) Bundle. e) TextWatcher. f) atividade. g) onCreate. h) R. i) R.drawable. j) R.string. k) findViewById.
- 3.2** a) Verdadeira. b) Falsa. A propriedade **weight** do *layout* de cada componente determina como ele deve ser dimensionado em relação aos outros componentes. c) Falsa. Os aplicativos Android *não têm* um método `main`. d) Falsa. Uma atividade ativa (ou em execução) é visível na tela e “tem o foco” – isto é, está no *primeiro plano*. e) Falsa. Uma atividade parada *não* é visível na tela e é provável que seja encerrada pelo sistema quando sua memória for necessária.

Exercícios

- 3.3** Preencha os espaços em branco em cada um dos seguintes enunciados:
- Strings literais devem ser colocadas no arquivo `strings.xml` da pasta _____ do aplicativo.
 - A _____ de um componente especifica sua importância relativa aos outros componentes.
 - A classe _____ do pacote `android.app` fornece os métodos de ciclo de vida básicos de um aplicativo.
 - A interface _____ do pacote `android.text` permite alterar o conteúdo e a marcação de texto em uma interface gráfica do usuário.
 - Você implementa a interface _____ do pacote `android.widget` para responder ao movimento do cursor de um elemento `SeekBar` feito pelo usuário.
 - Os aplicativos Android têm quatro tipos de componentes executáveis — atividades, serviços, provedores de conteúdo e _____.
 - Ao longo de toda sua vida, uma atividade pode estar em um dentre vários estados — ativa (ou em execução), pausada ou _____. A atividade transita entre esses estados em resposta a vários eventos.
 - Classe _____ (aninhada na classe `R`) — contém constantes para os componentes da interface gráfica do usuário em seus arquivos de `layout XML`.
 - O método `setContentView` usa uma constante recebida para carregar o documento XML correspondente, o qual é então analisado e convertido na interface gráfica do usuário do aplicativo. Esse processo é conhecido como _____ a interface gráfica do usuário.
- 3.4** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- Por padrão, o componente `SeekBar` permite selecionar valores de 0 a 255.
 - Um componente da interface gráfica do usuário pode abranger várias colunas em um elemento `GridLayout`.
 - Toda subclasse de `Activity` deve sobrescrever o método `Construct`.
 - Uma atividade pausada é visível na tela e tem o foco.
 - Inicializações demoradas devem ser feitas em um método `onCreate` e não no processo de segundo plano.
 - Você sobrescreve o método `onStart` para inicializar o aplicativo quando ele é iniciado.
- 3.5** (*Aplicativo Tip Calculator melhorado*) Faça as seguintes melhorias no aplicativo `Tip Calculator`:
- Acrescente uma opção para calcular a gorjeta com base no preço antes ou depois do imposto.

- b) Permita que o usuário digite o número de pessoas na festa. Calcule e exiba o valor devido por cada pessoa se a conta fosse dividida igualmente entre os participantes da festa.

- 3.6** (*Aplicativo Mortgage Calculator*) Crie um aplicativo de cálculo de hipoteca que permita ao usuário digitar o valor da compra, a entrada e uma taxa de juros. Com base nesses valores, o aplicativo deve calcular o valor do empréstimo (valor da compra menos a entrada) e exibir o pagamento mensal para empréstimos de 10, 20 e 30 anos. Permita que o usuário selecione uma duração personalizada para o empréstimo (em anos) usando um componente SeekBar e exiba o pagamento mensal para essa duração personalizada.
- 3.7** (*Aplicativo College Loan Payoff Calculator*) Um banco oferece crédito educativo que pode ser quitado em 5, 10, 15, 20, 25 ou 30 anos. Escreva um aplicativo que permita ao usuário digitar o valor do empréstimo e a taxa de juros anual. Com base nesses valores, o aplicativo deve exibir a duração do empréstimo em anos e seus pagamentos mensais correspondentes.
- 3.8** (*Aplicativo Car Payment Calculator*) Normalmente, os bancos oferecem empréstimos para compra de automóveis por períodos que variam de 2 a 5 anos (24 a 60 meses). Os mutuários quitam os empréstimos com prestações mensais. O valor de cada pagamento mensal é baseado na duração do empréstimo, no valor financiado e na taxa de juros. Crie um aplicativo que permita ao cliente digitar o preço de um carro, o valor da entrada e a taxa de juros anual do empréstimo. O aplicativo deve exibir a duração do empréstimo em meses e os pagamentos mensais para financiamentos de 3, 4 e 5 anos. A variedade de opções permite ao usuário comparar facilmente os planos de financiamento e escolher o mais adequado.
- 3.9** (*Aplicativo Miles-Per-Gallon Calculator*) Os motoristas frequentemente querem saber o consumo de seus carros para que possam estimar os gastos com combustível. Desenvolva um aplicativo que permita ao usuário digitar o número de quilômetros percorridos e o número de litros utilizados, e que calcule e exiba a quilometragem por litro correspondente.
- 3.10** (*Aplicativo Body Mass Index Calculator*) As fórmulas para calcular o índice de massa corporal (IMC) são

$$IMC = \frac{\text{pesoEmLibras} \times 703}{\text{alturaEmPolegadas} \times \text{alturaEmPolegadas}}$$

ou

$$IMC = \frac{\text{pesoEmQuilogramas}}{\text{alturaEmMetros} \times \text{alturaEmMetros}}$$

Crie um aplicativo de cálculo de IMC que permita aos usuários digitar seu peso e altura, e que identifique se estão digitando esses valores em unidades inglesas ou métricas, e então calcule e exiba o índice de massa corporal do usuário. O aplicativo também deve exibir as seguintes informações do Departamento de Saúde e Serviço Social/Institutos Nacionais de Saúde, para que o usuário possa avaliar seu IMC:

VALORES DE IMC
 Abaixo do peso: menos de 18.5
 Normal: entre 18.5 e 24.9
 Sobrepeso: entre 25 e 29.9
 Obeso: 30 ou mais

- 3.11** (*Aplicativo Target-Heart-Rate Calculator*) Ao fazer exercícios físicos, você pode usar um monitor de frequência de batimentos cardíacos para ver se seu batimento permanece dentro de uma faixa segura, sugerida por treinadores e médicos. De acordo com a American Heart Association (AHA), a fórmula para calcular seu *batimento cardíaco máximo* por minuto é

220 menos sua idade em anos (<http://bit.ly/AHATargetHeartRates>). Seus *batimentos cardíacos desejáveis* estão em um intervalo de 50 a 85% de seus batimentos cardíacos máximos. [Obs.: essas fórmulas são estimativas fornecidas pela AHA. Os batimentos cardíacos máximo e desejável podem variar de acordo com a saúde, capacidade física e sexo do indivíduo. Consulte sempre um médico ou profissional de saúde qualificado antes de iniciar ou modificar um programa de exercícios físicos.] Escreva um aplicativo que insira a idade da pessoa e, então, calcule e exiba os batimentos cardíacos máximos e desejáveis para essa pessoa.

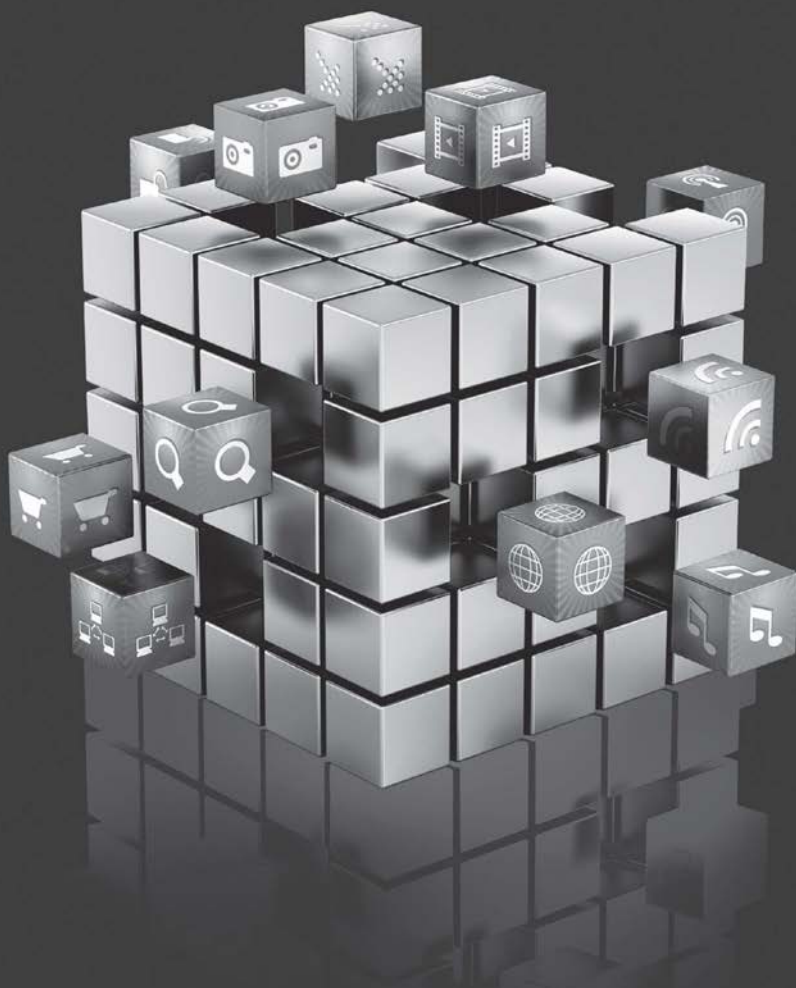
4

Aplicativo Twitter® Searches

Objetivos

Neste capítulo, você vai:

- Dar suporte para as orientações retrato e paisagem do dispositivo.
- Ampliar o componente `ListActivity` para criar uma atividade que exibe uma lista de itens em um componente `ListView`.
- Permitir que os usuários interajam com um aplicativo por meio de um componente `ImageButton`.
- Manipular coleções de dados.
- Usar `SharedPreferences` para armazenar pares chave-valor de dados associados a um aplicativo.
- Usar um componente `SharedPreferences.Editor` para modificar pares chave-valor de dados associados a um aplicativo.
- Usar um componente `ArrayAdapter` para especificar os dados de um componente `ListView`.
- Usar um objeto `AlertDialog.Builder` para criar caixas de diálogo que exibem opções, como elementos `Button` ou em um elemento `ListView`.
- Usar um objeto `Intent` implícito para abrir um site em um navegador.
- Usar um objeto `Intent` implícito para exibir um selecionador de intenção contendo uma lista de aplicativos que podem compartilhar texto.
- Ocultar o teclado virtual via programação.



4.1	Introdução	4.4.4	Barra de ferramentas do editor
4.2	Teste do aplicativo		Graphical Layout
4.2.1	Importação e execução do aplicativo	4.4.5	Layout do item ListView: list_
4.2.2	Adição de uma busca favorita		item.xml
4.2.3	Visualização dos resultados de uma	4.5	Construção da classe MainActivity
	busca no Twitter	4.5.1	As instruções package e import
4.2.4	Edição de uma pesquisa	4.5.2	Extensão de ListActivity
4.2.5	Compartilhamento de uma pesquisa	4.5.3	Campos da classe MainActivity
4.2.6	Exclusão de uma pesquisa	4.5.4	Sobrescrevendo o método onCreate
4.2.7	Rolagem por pesquisas salvas		de Activity
4.3	Visão geral das tecnologias	4.5.5	Classe interna anônima
4.3.1	ListView		que implementa a interface
4.3.2	ListActivity		OnClickListener de saveButton
4.3.3	Personalização do layout de um		para salvar uma pesquisa nova ou
	componente ListActivity		atualizada
4.3.4	ImageButton	4.5.6	Método addTaggedSearch
4.3.5	SharedPreferences	4.5.7	Classe interna anônima
4.3.6	Objetos Intent para ativar outras		que implementa a interface
	atividades		OnItemClickListener de ListView
4.3.7	AlertDialog		para exibir resultados de pesquisa
4.3.8	AndroidManifest.xml	4.5.8	Classe interna anônima
4.4	Construção da interface gráfica do		que implementa a interface
	usuário do aplicativo		OnItemLongClickListener de
4.4.1	Criação do projeto		ListView para compartilhar, editar
4.4.2	Visão geral de activity_main.xml		ou excluir uma pesquisa
4.4.3	Adição de GridLayout e	4.5.9	Método shareSearch
	componentes	4.5.10	Método deleteSearch
		4.6	AndroidManifest.xml
		4.7	Para finalizar

Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios

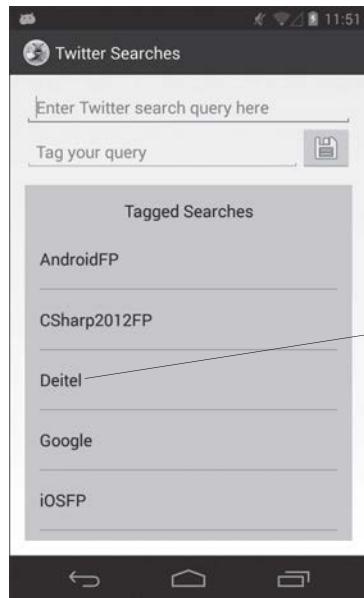
4.1 Introdução

O mecanismo de busca do Twitter facilita seguir os trending topics (assuntos do momento) que estão sendo discutidos por mais de 500 milhões de usuários do Twitter. As pesquisas podem ser aperfeiçoadas com os *operadores de busca* do Twitter (examinados na Seção 4.2), frequentemente resultando em strings de busca longas, demoradas e inconvenientes para digitar em um dispositivo móvel. O aplicativo **Twitter Searches** (Fig. 4.1) permite salvar suas consultas de busca favoritas com nomes identificadores curtos e fáceis de lembrar (Fig. 4.1(a)). Você pode então tocar em um nome identificador para seguir *tweets* sobre determinado tópico de forma rápida e fácil (Fig. 4.1(b)). Conforme verá, o aplicativo também permite *compartilhar*, *editar* e *excluir* pesquisas salvas.

O aplicativo aceita as orientações retrato e paisagem dos dispositivos. Em alguns aplicativos, você fará isso fornecendo layouts separados para cada orientação. Neste aplicativo, damos suporte para as duas orientações, projetando a interface gráfica do usuário de modo que ela possa ajustar dinamicamente o tamanho dos componentes da interface de acordo com a orientação atual.

Primeiramente, você vai testar o aplicativo. Em seguida, vamos ver um panorama das tecnologias utilizadas para construí-lo. Depois, você vai projetar a interface gráfica do usuário do aplicativo. Por fim, apresentaremos o código-fonte completo do aplicativo e o examinaremos, discutindo os novos recursos do aplicativo com mais detalhes.

a) Aplicativo com várias pesquisas salvas



Consulta de busca enviada para o Twitter

Toque em um identificador para fazer a consulta de busca correspondente

Toque em um identificador e mantenha o dedo na posição (o que é conhecido como pressionamento longo) para ver um componente AlertDialog com opções para compartilhar, editar ou excluir a pesquisa

b) Aplicativo depois que o usuário toca em "Deitel"

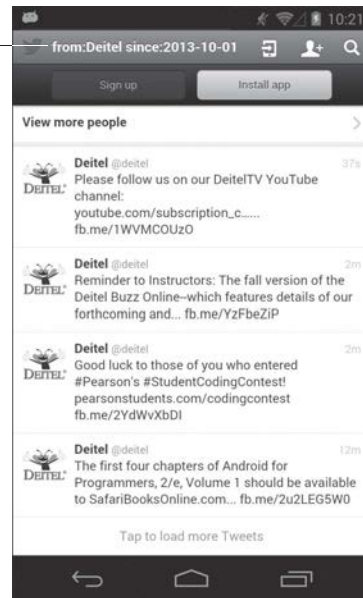


Figura 4.1 Aplicativo Twitter Searches.

4.2 Teste do aplicativo

Nesta seção, você vai testar o aplicativo **Twitter Searches**. Abra o IDE Android Developer Tools e importe o projeto do aplicativo **Twitter Searches**. Como fez no Capítulo 3, ative o AVD do Nexus 4 – ou conecte seu dispositivo Android ao computador – para que você possa testar o aplicativo. As capturas de tela mostradas neste capítulo foram tiradas em um telefone Nexus 4.

4.2.1 Importação e execução do aplicativo

Execute os passos a seguir para importar o aplicativo para o IDE:

1. **Abra a caixa de diálogo Import.** Selecione **File > Import...**
2. **Importe o projeto do aplicativo Twitter Searches.** Expanda o nó **General** e selecione **Existing Projects into Workspace**. Clique em **Next >** para continuar no passo **Import Projects**. Certifique-se de que **Select root directory** esteja selecionado e, em seguida, clique em **Browse...** Localize a pasta **TwitterSearches** na pasta de exemplos do livro, selecione-a e clique em **OK**. Certifique-se de que **Copy projects into workspace** *não* esteja selecionado. Clique em **Finish** a fim de importar o projeto para que ele apareça na janela **Package Explorer**.
3. **Ative o aplicativo Twitter Searches.** Clique com o botão direito do mouse no projeto **TwitterSearches** na janela **Package Explorer** e selecione **Run As > Android Application** para executar o aplicativo **Twitter Searches** no AVD. Isso constrói o projeto e executa o aplicativo (Fig. 4.2).

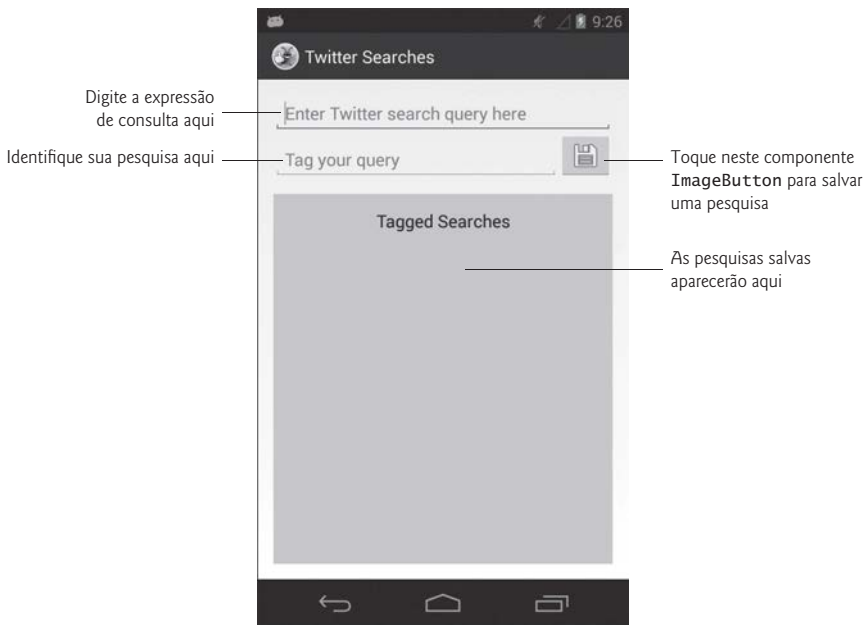


Figura 4.2 O aplicativo **Twitter Searches** ao ser executado pela primeira vez.

4.2.2 Adição de uma busca favorita

Toque no componente `EditText` superior e digite `from:deitel` como consulta de busca – o operador `from:` localiza *tweets* de uma conta do Twitter especificada. A Figura 4.3 mostra diversos operadores de pesquisa do Twitter – vários operadores podem ser usados em conjunto para construir consultas mais complexas. Uma lista completa pode ser encontrada em

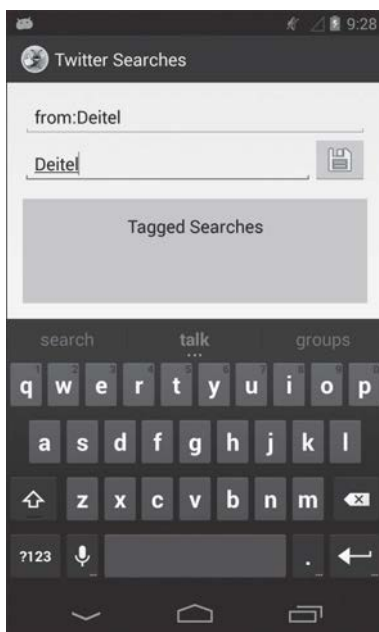
<http://bit.ly/TwitterSearchOperators>

Exemplo	Localiza <i>tweets</i> contendo
<code>deitel iOS6</code>	Operador <i>and</i> lógico implícito – Localiza <i>tweets</i> contendo <code>deitel</code> e <code>iOS6</code> .
<code>deitel OR iOS6</code>	Operador <i>OR</i> lógico – Localiza <i>tweets</i> contendo <code>deitel</code> ou <code>iOS6</code> ou ambos.
<code>"how to program"</code>	String entre aspas ("") – Localiza <i>tweets</i> contendo a frase exata "how to program".
<code>deitel ?</code>	? (ponto de interrogação) – Localiza <i>tweets</i> que fazem perguntas sobre <code>deitel</code> .
<code>deitel -sue</code>	- (sinal de subtração) – Localiza <i>tweets</i> contendo <code>deitel</code> , mas não <code>sue</code> .
<code>deitel :)</code>	:) (face alegre) – Localiza <i>tweets</i> de <i>atitude positiva</i> contendo <code>deitel</code> .
<code>deitel :(</code>	: ((face triste) – Localiza <i>tweets</i> de <i>atitude negativa</i> contendo <code>deitel</code> .
<code>since:2013-10-01</code>	Localiza <i>tweets</i> que ocorreram na data especificada ou depois dela, a qual deve estar na forma AAAA-MM-DD.
<code>near:"New York City"</code>	Localiza <i>tweets</i> que foram enviados perto de "New York City".
<code>from:deitel</code>	Localiza <i>tweets</i> da conta @deitel no Twitter.
<code>to:deitel</code>	Localiza <i>tweets</i> para a conta @deitel no Twitter.

Figura 4.3 Alguns operadores de pesquisa do Twitter.

No componente `EditText` inferior, digite `Deitel` como identificador para a consulta de busca (Fig. 4.4(a)). Esse vai ser o *nome curto* exibido em uma lista na seção **Tagged Searches** do aplicativo. Toque no botão *salvar* (📁) para salvar a pesquisa – o identificador “Deitel” aparece na lista, sob o cabeçalho **Tagged Searches** (Fig. 4.4(b)). Quando uma pesquisa é salva, o teclado virtual desaparece para que você possa ver sua lista de pesquisas salvas – você vai aprender a ocultar o teclado virtual por meio de programação na Seção 4.5.5.

a) Digitando uma pesquisa e um identificador de pesquisa do Twitter



b) O aplicativo depois de salvar a pesquisa e o identificador

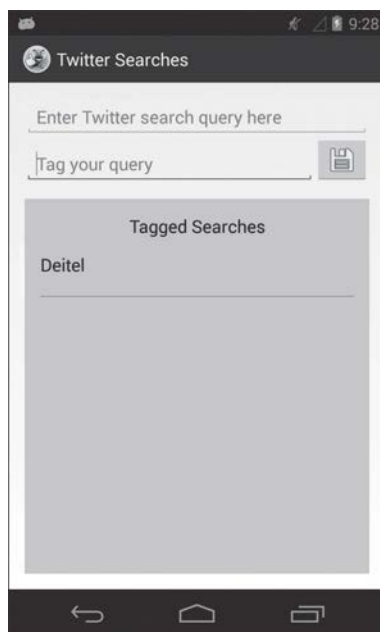


Figura 4.4 Inserindo uma pesquisa do Twitter.

4.2.3 Visualização dos resultados de uma busca no Twitter

Para ver os resultados da pesquisa, toque no identificador “Deitel”. Isso ativa o navegador Web do dispositivo e passa uma URL que representa a pesquisa salva para o site do Twitter. O Twitter obtém a consulta de busca a partir da URL e retorna os *tweets* correspondentes à consulta (se houver) como uma página Web. Então, o navegador Web exibe essa página de resultados (Fig. 4.5). Quando terminar de ver os resultados, toque no botão *voltar* (←) para voltar ao aplicativo **Twitter Searches**, em que você pode salvar mais pesquisas e editar, excluir e compartilhar pesquisas salvas anteriormente.

4.2.4 Edição de uma pesquisa

Também é possível *compartilhar*, *editar* ou *excluir* uma pesquisa. Para ver essas opções, faça um *pressionamento longo* no identificador da pesquisa – isto é, toque no identificador e mantenha o dedo na tela. Para fazer um pressionamento longo usando um AVD, clique e mantenha o botão esquerdo do mouse pressionado no identificador da pesquisa.

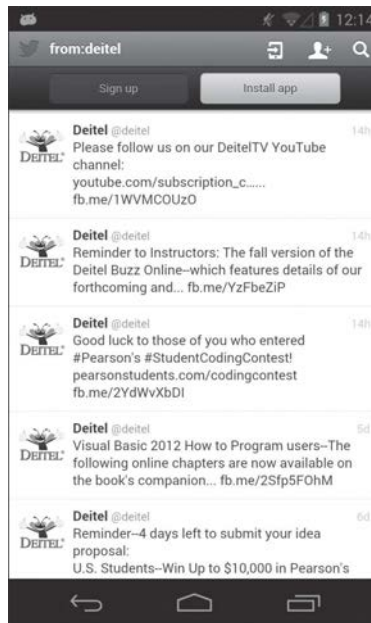


Figura 4.5 Vendo resultados de pesquisa.

Quando um pressionamento longo é feito em “Deitel”, o componente AlertDialog na Fig. 4.6(a) exibe as opções **Share**, **Edit** e **Delete** para a pesquisa identificada como “Deitel”. Se não quiser executar essas tarefas, toque em **Cancel**.

- a) Selecionando **Edit** para editar uma pesquisa já existente b) Editando a pesquisa “Deitel” salva

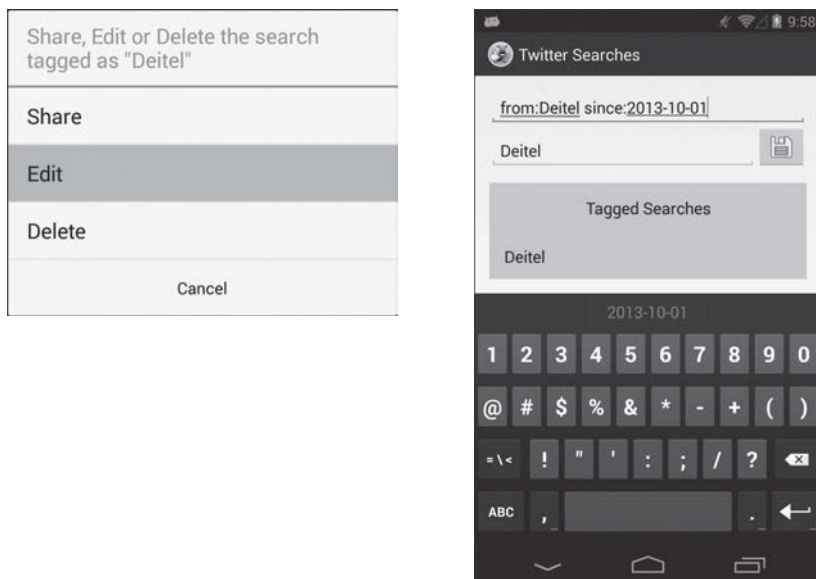


Figura 4.6 Editando uma pesquisa salva.

Para editar a pesquisa identificada como “Deitel”, toque na opção **Edit** da caixa de diálogo. O aplicativo carrega a consulta e o identificador da pesquisa nos componentes `EditText` para edição. Vamos restringir nossa busca a *tweets* a partir de 1º de outubro de 2013, adicionando `since:2013-10-01` ao final da consulta (Fig. 4.6(b)) na parte superior do componente `EditText`. O operador `since:` restringe os resultados da pesquisa aos *tweets* que ocorreram na data especificada ou depois dela (na forma `aaaa-mm-dd`). Toque no botão *salvar* (💾) para atualizar a pesquisa salva e, então, veja os resultados atualizados (Fig. 4.7) tocando em **Deitel** na seção **Tagged Searches** do aplicativo. [Obs.: Alterar o nome do identificador vai criar uma *nova* pesquisa – isso é útil se você quiser basear uma nova consulta em outra salva anteriormente.]

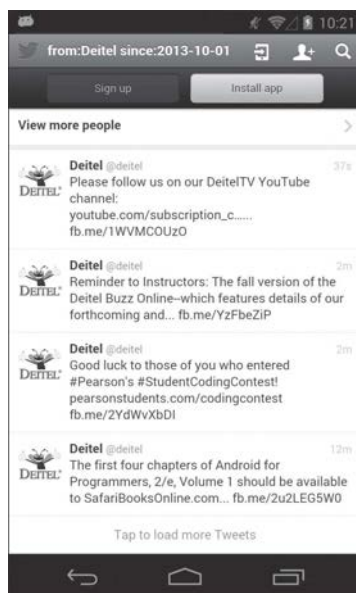


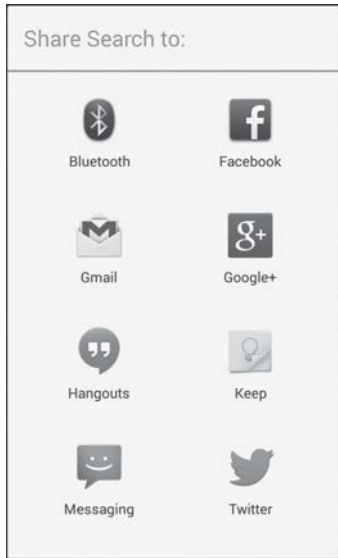
Figura 4.7 Vendo os resultados atualizados da pesquisa “Deitel”.

4.2.5 Compartilhamento de uma pesquisa

O Android facilita compartilhar vários tipos de informação de um aplicativo via e-mail, troca de mensagens instantânea (SMS), Facebook, Google+, Twitter e muito mais. Neste aplicativo, você pode compartilhar uma pesquisa favorita fazendo um *pressionamento longo* no identificador da pesquisa e selecionando **Share** no componente `AlertDialog` que aparece. Isso exibe o assim chamado *selecionador de intenção* (Fig. 4.8(a)), o qual pode variar de acordo com o tipo de conteúdo que você está compartilhando e com os aplicativos que podem manipular esse conteúdo. Neste aplicativo, estamos compartilhando texto, e o selecionador de intenção de nosso telefone (não o AVD) mostra aplicativos capazes de manipular texto, como **Facebook**, **Gmail**, **Google+**, **Messaging** (troca de mensagens instantâneas) e **Twitter**. Se nenhum aplicativo puder manipular o conteúdo, o selecionador de intenção exibirá uma mensagem informando isso. Se somente um aplicativo puder manipular o conteúdo, ele será ativado sem que você precise escolher o aplicativo a ser usado no selecionador de intenção. A Figura 4.8(b) mostra a tela **Compose** do aplicativo Gmail com o assunto e o corpo do e-mail preenchidos. O Gmail mostra também

seu endereço de e-mail acima do campo **To** (excluímos o endereço de e-mail na captura de tela por questões de privacidade).

a) Selecionador de intenção mostrando opções de compartilhamento



b) Tela **Compose** do aplicativo Gmail para um e-mail contendo a pesquisa "Deitel"

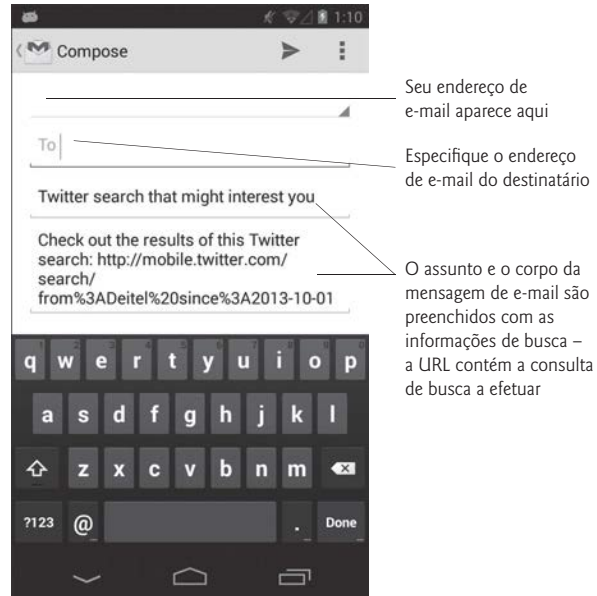


Figura 4.8 Compartilhando uma pesquisa via e-mail.

4.2.6 Exclusão de uma pesquisa

Para excluir uma pesquisa, faça um *pressionamento longo* no identificador da pesquisa e selecione **Delete** no componente `AlertDialog` que aparece. O aplicativo pede para você confirmar que deseja excluir a pesquisa (Fig. 4.9) – tocar em **Cancel** o leva de volta à tela principal *sem* excluir a pesquisa. Tocar em **Delete** exclui a pesquisa.

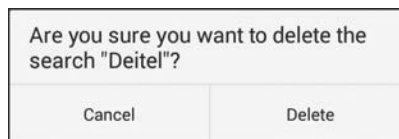


Figura 4.9 Componente `AlertDialog` confirmando uma exclusão.

4.2.7 Rolagem por pesquisas salvas

A Figura 4.10 mostra o aplicativo depois de salvarmos 10 pesquisas favoritas – apenas cinco das quais estão visíveis. O aplicativo permite rolar por suas pesquisas favoritas caso exista mais do que pode aparecer na tela simultaneamente. O componente da interface gráfica do usuário que exibe a lista de pesquisas é um `ListView` (discutido na Seção 4.3.1). Para rolar, *arraste* ou dê um *toque* com o dedo (ou com o mouse, em um AVD)

para cima ou para baixo na lista de **Tagged Searches**. Além disso, gire o dispositivo para a orientação *paisagem* a fim de ver a interface gráfica se ajustar dinamicamente.

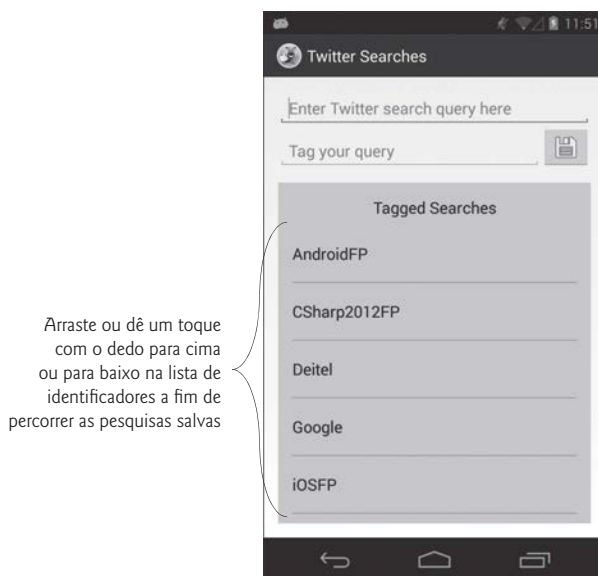


Figura 4.10 O aplicativo com mais pesquisas do que podem ser exibidas na tela.

4.3 Visão geral das tecnologias

Esta seção apresenta os recursos que você vai usar para construir o aplicativo **Twitter Searches**.

4.3.1 ListView

Muitos aplicativos móveis exibem listas de informação. Por exemplo, um aplicativo de e-mail exibe uma lista de novos e-mails, um aplicativo de agenda eletrônica exibe uma lista de contatos, um aplicativo de notícias exibe uma lista de manchetes, etc. Em cada caso, o usuário toca em um item da lista para ver mais informações – por exemplo, o conteúdo do e-mail selecionado, os detalhes do contato selecionado ou o texto da matéria da notícia selecionada. Este aplicativo usa um componente **ListView** (pacote `android.widget`) para exibir uma lista de pesquisas identificadas que pode ser *rolada* caso a lista completa não possa ser exibida na tela. Você pode especificar como vai formatar cada item de **ListView**. Para este aplicativo, exibiremos o identificador de cada pesquisa como um objeto `String` em um componente `TextView`. Em aplicativos posteriores, você vai personalizar completamente o conteúdo exibido para cada item de **ListView** – exibindo imagens, texto e componentes `Button`.

4.3.2 ListActivity

Quando a tarefa principal de uma atividade é exibir uma lista de itens *rolante*, você pode estender a classe **ListActivity** (pacote `android.app`), a qual usa um componente **ListView** que ocupa o aplicativo inteiro como layout padrão. **ListView** é uma subclasse de **Adapter-**

View (pacote `android.widget`) – um componente de interface gráfica do usuário é vinculado a uma fonte de dados por meio de um objeto **Adapter** (pacote `android.widget`). Neste aplicativo, usamos um componente **ArrayAdapter** (pacote `android.widget`) para criar um objeto que preenche o elemento `ListView` utilizando dados de um objeto coleção `ArrayList`. Isso é conhecido como **vinculação de dados**. Vários tipos de componentes `AdapterView` podem ser vinculados a dados usando um objeto `Adapter`. No Capítulo 8, você vai aprender a vincular dados de banco de dados a um componente `ListView`. Para obter mais detalhes sobre vinculação de dados no Android e vários tutoriais, visite:

<http://developer.android.com/guide/topics/ui/binding.html>

4.3.3 Personalização do layout de um componente `ListActivity`

A interface gráfica do usuário padrão de um componente `ListActivity` contém apenas um elemento `ListView` que preenche a área cliente da tela entre as barras de sistema superior e inferior do Android (as quais foram explicadas na Fig. 2.1). Se a interface gráfica de um componente `ListActivity` exige apenas o elemento `ListView` padrão, você *não* precisa definir um layout separado para sua subclasse `ListActivity`.

O componente `MainActivity` do aplicativo **Twitter Searches** exibe vários componentes de interface gráfica do usuário. Por isso, você vai definir um layout *personalizado* para `MainActivity`. Ao personalizar a interface gráfica do usuário de uma subclasse de `ListActivity`, o layout *deve* conter um componente `ListView` com o atributo `id` configurado como `"@android:id/list"` – o nome utilizado pela classe `ListActivity` para fazer referência a seu componente `ListView`.

4.3.4 `ImageButton`

Os usuários frequentemente tocam em botões para iniciar ações em um programa. Para salvar um par consulta-identificador de uma pesquisa neste aplicativo, você toca em um componente **`ImageButton`** (pacote `android.widget`). `ImageButton` é uma subclasse de `ImageView`, a qual fornece recursos adicionais que permitem utilizar uma imagem, como um objeto `Button`, (pacote `android.widget`) para iniciar uma ação.

4.3.5 `SharedPreferences`

É possível ter um ou mais arquivos contendo pares *chave-valor* associados a cada aplicativo – cada *chave* permite pesquisar rapidamente um *valor* correspondente. Usamos esse recurso para manipular um arquivo chamado `searches`, no qual armazenamos os pares de identificadores (as *chaves*) e consultas de busca no Twitter (os *valores*) criadas pelo usuário. Para ler os pares chave-valor desse arquivo, vamos usar objetos **`SharedPreferences`** (pacote `android.content`). Para modificar o conteúdo do arquivo, vamos usar objetos **`SharedPreferences.Editor`** (pacote `android.content`). As chaves presentes no arquivo devem ser objetos `String` e os valores podem ser objetos `String` ou valores de tipos primitivos.

Este aplicativo lê as pesquisas salvas no método `onCreate` de `Activity` – isso é aceitável apenas porque o volume de dados que está sendo carregado é pequeno. Quando um aplicativo é ativado, o Android cria uma thread principal, chamada thread UI, que manipula todas as interações da interface gráfica do usuário. Todo processamento de interface gráfica do usuário deve ser feito nessa thread. *Operações de entrada/saída extensas, como carregamento de dados de arquivos e bancos de dados, não devem ser efetuadas na thread UI, pois podem afetar a rapidez de resposta de seu aplicativo.* Em capítulos posteriores, vamos mostrar como fazer E/S em threads separadas.

4.3.6 Objetos Intent para ativar outras atividades

O Android usa uma técnica conhecida como **troca de mensagens de intenção** para transmitir informações entre atividades dentro de um aplicativo ou atividades em aplicativos separados. Cada atividade pode especificar **filtros de intenção** indicando as *ações* que consegue manipular. Os filtros de intenção são definidos no arquivo `AndroidManifest.xml`. Na verdade, em cada aplicativo até aqui, o IDE criou um filtro de intenção para sua única atividade, indicando que ele podia responder à ação predefinida chamada `android.intent.action.MAIN`, a qual especifica que a atividade pode ser usada para *ativar* o aplicativo para iniciar sua execução.

Um objeto **Intent** é usado para ativar uma atividade – ele indica uma *ação* a ser executada e os *dados* nos quais essa ação vai atuar. Neste aplicativo, quando o usuário toca em um identificador de pesquisa, criamos uma URL contendo a consulta de busca no Twitter. Carregamos a URL em um navegador Web criando um novo elemento Intent para ver uma URL, passando então esse Intent para o **método startActivity**, o qual nosso aplicativo herda indiretamente da classe Activity. Para ver uma URL, `startActivity` ativa o navegador Web do dispositivo para exibir o conteúdo – neste aplicativo, os resultados de uma busca no Twitter.

Usamos também um elemento Intent e o método `startActivity` para exibir um **selecionador de intenção** – uma interface gráfica do usuário que mostra uma lista de aplicativos que podem manipular o objeto Intent especificado. Usamos isso ao compartilhar uma pesquisa salva para permitir ao usuário escolher como vai compartilhar uma busca.

Objetos Intent implícitos e explícitos

Os objetos Intent usados neste aplicativo são **implícitos** – *não especificaremos um componente para exibir a página Web; em vez disso, permitiremos que o Android ative a atividade mais adequada com base no tipo dos dados*. Se várias atividades puderem tratar da ação e dos dados passados para `startActivity`, o sistema exibirá uma *caixa de diálogo* na qual o usuário poderá selecionar a atividade a ser usada. Se o sistema não consegue encontrar uma atividade para tratar da ação, então o método `startActivity` lança uma exceção `ActivityNotFoundException`. Em geral, considera-se uma boa prática tratar essa exceção. Optamos por não fazer isso neste aplicativo, pois os dispositivos Android nos quais esse aplicativo provavelmente vai ser instalado têm um navegador capaz de exibir uma página Web. Em aplicativos futuros, também usaremos objetos **Intent explícitos**, os quais indicam precisamente a atividade a ser iniciada. Para obter mais informações sobre objetos Intent, visite:

<http://developer.android.com/guide/components/intents-filters.html>

4.3.7 AlertDialog

As mensagens, opções e confirmações podem ser exibidas para os usuários do aplicativo por meio de componentes **AlertDialog**. Enquanto uma caixa de diálogo está sendo exibida, o usuário não pode interagir com o aplicativo – isso é conhecido como **caixa de diálogo modal**. Conforme você vai ver, as configurações da caixa de diálogo são especificadas com um objeto **AlertDialog.Builder**, sendo ele então utilizado para criar o componente `AlertDialog`.

Os componentes `AlertDialog` podem exibir botões, caixas de seleção, botões de opção e listas de itens em que o usuário pode tocar para responder à mensagem da caixa de diálogo. Um componente `AlertDialog` padrão pode ter até três botões representando:

- Uma ação *negativa* – Cancela a ação especificada da caixa de diálogo, frequentemente rotulada com **Cancel** ou **No**. Esse é o botão mais à esquerda quando existem vários na caixa de diálogo.
- Uma ação *positiva* – Aceita a ação especificada da caixa de diálogo, frequentemente rotulada com **OK** ou **Yes**. Esse é o botão mais à direita quando existem vários na caixa de diálogo.
- Uma ação *neutra* – Esse botão indica que o usuário não quer cancelar nem aceitar a ação especificada pela caixa de diálogo. Por exemplo, um aplicativo que pede ao usuário para que se registre a fim de obter acesso a recursos adicionais poderia fornecer o botão neutro **Remind Me Later** (Lembre-me depois).

Neste aplicativo, usamos componentes `AlertDialog` com várias finalidades:

- Para exibir uma mensagem ao usuário, caso nenhum ou ambos os componentes `EditText` de consulta e identificação estejam vazios. Essa caixa de diálogo conterá somente um botão positivo.
- Para exibir as opções **Share**, **Edit** e **Delete** para uma pesquisa. Essa caixa de diálogo conterá uma lista de opções e um botão negativo.
- Para que o usuário confirme antes de excluir uma pesquisa – no caso de ter tocado acidentalmente na opção **Delete** para uma pesquisa.

Mais informações sobre as caixas de diálogo do Android podem ser encontradas em:

<http://developer.android.com/guide/topics/ui/dialogs.html>

4.3.8 AndroidManifest.xml

Conforme você aprendeu no Capítulo 3, o arquivo `AndroidManifest.xml` é gerado no momento em que um aplicativo é criado. Para este aplicativo, vamos mostrar a você como adicionar no manifesto uma configuração que impede a exibição do teclado virtual quando o aplicativo é carregado pela primeira vez. Para ver os detalhes completos de `AndroidManifest.xml`, visite:

<http://developer.android.com/guide/topics/manifest/manifest-intro.html>

Vamos abordar vários aspectos do arquivo `AndroidManifest.xml` por todo o livro.

4.4 Construção da interface gráfica do usuário do aplicativo

Nesta seção, vamos construir a interface gráfica do usuário do aplicativo **Twitter Searches**. Também vamos criar um segundo layout XML, o qual o componente `ListView` vai inflar dinamicamente e usar para exibir cada item.

4.4.1 Criação do projeto

Lembre-se de que o IDE Android Developer Tools só permite *um* projeto com determinado nome por área de trabalho; portanto, antes de criar o novo projeto, exclua o projeto `TwitterSearches` que você testou na Seção 4.2. Para isso, clique nele com o botão direito do mouse e selecione **Delete**. Na caixa de diálogo que aparece, certifique-se de que **Delete project contents on disk** *não* esteja selecionado e, em seguida, clique em **OK**. Isso remove o projeto da área de trabalho, mas deixa a pasta e os arquivos do projeto no disco para o caso de posteriormente você querer ver o aplicativo original outra vez.

Criando um novo projeto Blank App

Crie um novo Android Application Project. Especifique os seguintes valores no primeiro passo de New Android Application da caixa de diálogo New Android Project e, em seguida, pressione Next >:

- Application name: Twitter Searches
- Project name: TwitterSearches
- Package name: com.deitel.twittersearches
- Minimum Required SDK: API18: Android 4.3
- Target SDK: API19: Android 4.4
- Compile With: API19: Android 4.4
- Theme: Holo Light with Dark Action Bar

No segundo passo de New Android Application da caixa de diálogo New Android Project, deixe as configurações padrão e pressione Next >. No passo Configure Launcher Icon, clique no botão Browse..., selecione uma imagem de ícone de aplicativo (fornecida na pasta images com os exemplos do livro), pressione Open e depois Next >. No passo Create Activity, selecione Blank Activity e pressione Next >. No passo Blank Activity, deixe as configurações padrão e clique em Finish para criar o projeto. No editor Graphical Layout, abra activity_main.xml e selecione Nexus 4 na lista suspensa de tipo de tela (como na Fig. 2.12). Mais uma vez, usaremos esse dispositivo como base para nosso projeto.

4.4.2 Visão geral de activity_main.xml

Como no Capítulo 3, o layout activity_main.xml deste aplicativo utiliza um componente GridLayout (Fig. 4.11). Neste aplicativo, o componente GridLayout contém três linhas e uma coluna. A Figura 4.12 mostra os nomes dos componentes da interface gráfica do aplicativo.

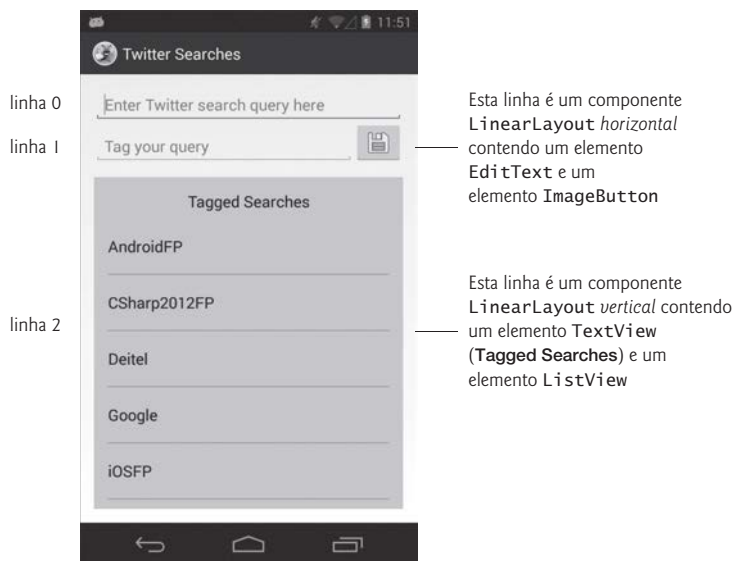


Figura 4.11 Linhas e colunas no componente GridLayout do aplicativo Twitter Searches.

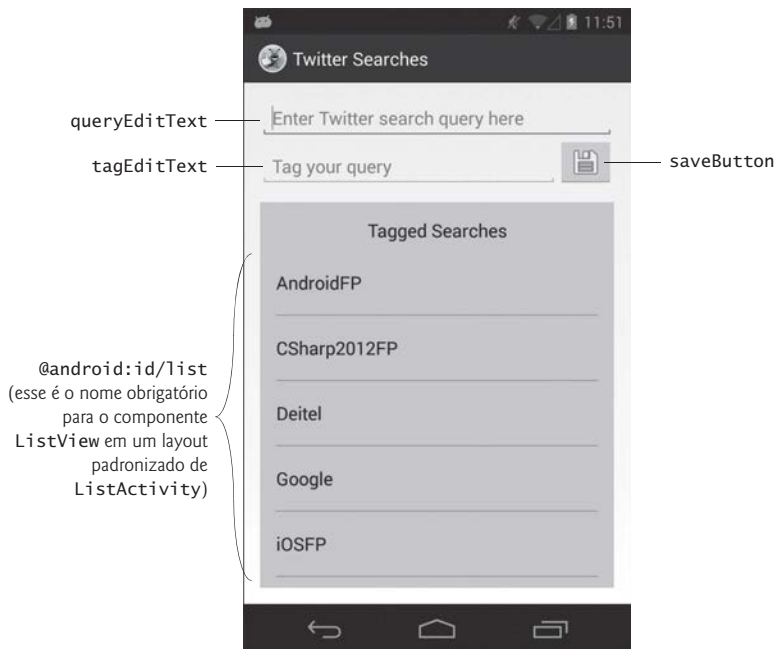


Figura 4.12 Componentes da interface gráfica do usuário do aplicativo **Twitter Searches** rotulados com seus valores de propriedade **Id**.

4.4.3 Adição de `GridLayout` e componentes

Utilizando as técnicas aprendidas no Capítulo 3, você vai construir a interface gráfica do usuário das Figuras 4.11 e 4.12. Todos os passos nas subseções a seguir presumem que você está trabalhando com o layout no editor **Graphical Layout** do IDE. Como lembrete, frequentemente é mais fácil selecionar um componente de interface gráfica do usuário em particular na janela **Outline**.

Você vai começar com o layout e os controles básicos e, em seguida, vai personalizar as propriedades dos controles para concluir o projeto. Use a janela **Outline** para adicionar elementos às linhas corretas do componente `GridLayout`. À medida que adicionar componentes de interface gráfica do usuário, configure suas propriedades **Id** como mostrado na Fig. 4.12 – nesse layout existem vários componentes que não exigem **Id**, pois nunca são referenciados no código Java do aplicativo. Além disso, lembre-se de definir todos os valores de string literais no arquivo `strings.xml` (localizado dentro da pasta `res/values` do aplicativo).

Passo 1: Mudando de `RelativeLayout` para `GridLayout`

Siga os passos da Seção 3.4.3 para trocar de `RelativeLayout` para `GridLayout`.

Passo 2: Configurando o componente `GridLayout`

Na janela **Outline**, selecione o componente `GridLayout` e configure as seguintes propriedades – para cada propriedade que estiver aninhada em um nó dentro da janela **Properties**, especificamos o nome do nó entre parênteses após o nome da propriedade:

- **Id:** `@+id/gridLayout`

- **Column Count** (nó **GridLayout**): 1 – Cada componente da interface gráfica do usuário aninhado diretamente no componente **GridLayout** será adicionado como uma nova linha.

O componente **GridLayout** preenche a área cliente inteira da tela, pois as propriedades **Width** e **Height** do layout (na seção **Layout Parameters** da janela **Properties**) são configuradas como **match_parent** pelo IDE.

Por padrão, o IDE configura as propriedades **Padding Left** e **Padding Right** como **@dimen/activity_horizontal_margin** – um recurso de dimensão predefinido no arquivo **dimens.xml** da pasta **res/values** do projeto. O valor desse recurso é 16dp; portanto, haverá um espaço de 16dp à esquerda e à direita do componente **GridLayout**. O IDE gerou esse recurso quando você criou o projeto do aplicativo. Do mesmo modo, o IDE configura as propriedades **Padding Top** e **Padding Bottom** como **@dimen/activity_vertical_margin** – outro recurso de dimensão predefinido com o valor 16dp. Portanto, haverá um espaço de 16dp acima e abaixo do componente **GridLayout**.



Observação sobre aparência e comportamento 4.1

De acordo com as diretrizes de projeto do Android, 16dp é o espaço recomendado entre as margens da área em que se pode tocar na tela de um dispositivo e o conteúdo do aplicativo; contudo, muitos aplicativos (como os jogos) utilizam a tela inteira.

Passo 3: Criando a primeira linha do componente **GridLayout**

Essa linha contém apenas um componente **EditText**. Arraste um componente **Plain Text** da seção **Text Fields** da **Palette** para o componente **GridLayout** na janela **Outline** e, então, configure sua propriedade **Id** como **@+id/queryEditText**. Na janela **Properties**, sob o nó **TextView**, exclua o valor da propriedade **Ems**, que não é utilizado neste aplicativo. Então, use a janela **Properties** para configurar as seguintes propriedades:

- **Width** (nó **Layout Parameters**): **wrap_content**
- **Height** (nó **Layout Parameters**): **wrap_content**
- **Gravity** (nó **Layout Parameters**): **fill_horizontal** – Isso garante que, quando o usuário girar o dispositivo, o componente **queryEditText** preencherá todo o espaço horizontal disponível. Usamos configurações de **Gravity** semelhantes para outros componentes da interface gráfica do usuário a fim de dar suporte para as orientações retrato e paisagem para a interface deste aplicativo.
- **Hint**: **@string/queryPrompt** – Crie um recurso **String**, como você fez em aplicativos anteriores, e dê a ele o valor "Enter Twitter search query here". Esse atributo exibe uma dica em um componente **EditText** *vazio*, a qual ajuda o usuário a entender o objetivo do componente. Esse texto também é falado pelo aplicativo **TalkBack** do Android para usuários deficientes visuais; portanto, fornecer dicas nos componentes **EditText** torna seu aplicativo mais acessível.



Observação sobre aparência e comportamento 4.2

As diretrizes de projeto do Android indicam que o texto exibido em sua interface gráfica do usuário deve ser breve, simples e amigável, com as palavras importantes em primeiro lugar. Para ver detalhes sobre o estilo de redação recomendada, consulte <http://developer.android.com/design/style/writing.html>.

- **IME Options** (nó **TextView**): **actionNext** – Este valor indica que o teclado do componente **queryEditText** conterá um botão **Next** em que o usuário pode tocar a fim de

mover o foco de entrada para o próximo componente de entrada (isto é, `tagEditText`, neste aplicativo). Isso facilita para o usuário preencher vários componentes de entrada em um formulário. Quando o próximo componente é outro `EditText`, o teclado apropriado é exibido sem que o usuário precise tocar no componente para passar o foco a ele.

Passo 4: Criando a segunda linha do componente `GridLayout`

Esta linha é um componente `LinearLayout` horizontal contendo um elemento `EditText` e um elemento `ImageButton`. Execute as tarefas a seguir para construir a interface gráfica do usuário da linha:

1. Arraste um componente **LinearLayout (Horizontal)** da seção **Layouts** da **Palette** para o componente `GridLayout` na janela **Outline**.
2. Arraste um componente **Plain Text** da seção **Text Fields** da **Palette** para o componente `LinearLayout` e, então, configure a propriedade `Id` como `@+id/tagEditText`.
3. Arraste um componente **ImageButton** da seção **Images & Media** da **Palette** para o componente `LinearLayout`. Isso exibe a caixa de diálogo **Resource Chooser** (Fig. 4.13) para que você possa escolher a imagem do botão. Por padrão, o botão de opção **Project Resources** da caixa de diálogo é selecionado para que você possa escolher imagens dos recursos do projeto (essas imagens seriam armazenadas em várias pastas `res/drawable` de seu projeto). Neste aplicativo, usamos o ícone de salvar padrão do Android (mostrado no lado direito da Fig. 4.13). Para isso, clique no botão de opção **System Resources**, selecione `ic_menu_save` e clique em **OK**. Em seguida, configure a propriedade `Id` como `@+id/saveButton`.

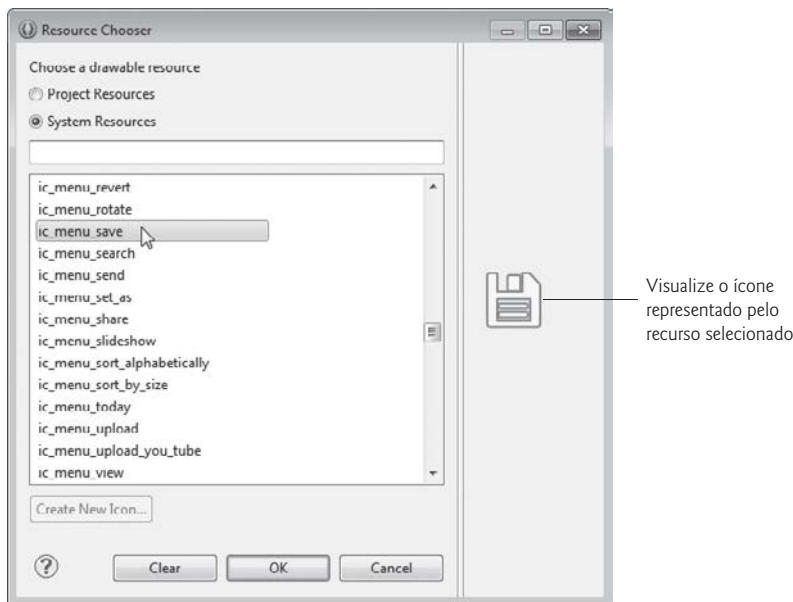


Figura 4.13 Caixa de diálogo **Resource Chooser**.

Com `tagEditText` selecionado, remova o valor da propriedade `Ems` do nó `TextView` na janela **Properties**. Em seguida, configure as seguintes propriedades:

- **Width** (nó **Layout Parameters**): `0dp` – O IDE recomenda esse valor quando a propriedade **Weight** também é configurada, para que possa organizar os componentes de forma mais eficiente.
- **Height** (nó **Layout Parameters**): `wrap_content`
- **Gravity** (nó **Layout Parameters**): `bottom|fill_horizontal` – Isso alinha a parte inferior do componente `tagEditText` com a parte inferior do componente `saveButton` e indica que `tagEditText` deve preencher o espaço horizontal disponível.
- **Weight** (nó **Layout Parameters**): `1` – Isso torna `tagEditText` mais importante do que `saveButton` nessa linha. Quando o Android organizar a linha, o componente `saveButton` ocupará apenas o espaço necessário, e o componente `tagEditText` ocupará todo o espaço horizontal restante.
- **Hint**: `@string/tagPrompt` – Crie um recurso `String` com o valor "Tag your query".
- **IME Options** (nó **TextView**): `actionDone` – Esse valor indica que o teclado do componente `queryEditText` conterá um botão **Done**, no qual o usuário pode tocar para remover o teclado da tela.

Com `saveButton` selecionado, apague o valor da propriedade **Weight** (nó **Layout Parameters**) e, então, configure as seguintes propriedades:

- **Width** (nó **Layout Parameters**): `wrap_content`
- **Height** (nó **Layout Parameters**): `wrap_content`
- **Content Description**: `@string/saveDescription` – Crie um recurso de `string` com o valor "Touch this button to save your tagged search".



Observação sobre aparência e comportamento 4.3

Lembre-se de que, no Android, considera-se a melhor prática garantir que todo componente da interface gráfica do usuário possa ser usado com `TalkBack`. Para componentes que não têm texto descritivo, como `ImageButton`, forneça texto para sua propriedade `Content Description`.

Passo 5: Criando a terceira linha do componente `GridLayout`

Esta linha é um componente `LinearLayout` vertical contendo um elemento `TextView` e um elemento `ListView`. Execute as tarefas a seguir para construir a interface gráfica do usuário da linha:

1. Arraste um componente **LinearLayout (Vertical)** da seção **Layouts** da **Palette** para o componente `GridLayout` na janela **Outline**.
2. Arraste um componente **Medium Text** da seção **Form Widgets** da **Palette** para o componente `LinearLayout`. Isso cria um componente `TextView` previamente configurado para exibir texto na fonte de tamanho médio do tema.
3. Arraste um componente **ListView** da seção **Composite** da **Palette** para o componente `LinearLayout` e configure a propriedade `id` como `@android:id/list` – lembre-se de que esse é o `id` exigido para `ListView` no layout personalizado de um componente `ListActivity`.

Com o componente `LinearLayout` vertical selecionado, configure as seguintes propriedades:

- **Height** (nó **Layout Parameters**): `0dp` – A altura real é determinada pela propriedade `Gravity`.

- **Gravity** (nó **Layout Parameters**): `fill` – Isso faz que o componente `LinearLayout` preencha todo o espaço horizontal e vertical disponível.
- **Top** (localizada no nó **Margins** do nó **Layout Parameters**): `@dimen/activity_vertical_margin` – Isso separa a parte superior do componente `LinearLayout` vertical do componente `LinearLayout` horizontal na segunda linha da interface gráfica do usuário.
- **Background** (nó **View**): `@android:color/holo_blue_bright` – Esse é um dos recursos de cor predefinidos no tema Android do aplicativo.
- **Padding Left/Right** (nó **View**): `@dimen/activity_horizontal_margin` – Isso garante que os componentes do elemento `LinearLayout` vertical sejam inseridos com uma distância de 16dp a partir das margens esquerda e direita do layout.
- **Padding Top** (nó **View**): `@dimen/activity_vertical_margin` – Isso garante que o componente superior dentro do elemento `LinearLayout` vertical seja inserido com uma distância de 16dp a partir da margem superior do layout.

Com o componente `TextView` vertical selecionado, configure as seguintes propriedades:

- **Width** (nó **Layout Parameters**): `match_parent`
- **Height** (nó **Layout Parameters**): `wrap_content`
- **Gravity** (nó **Layout Parameters**): `fill_horizontal` – Isso faz o componente `TextView` preencher a largura do componente `LinearLayout` vertical (menos o *padding* no layout).
- **Gravity** (nó **TextView**): `center_horizontal` – Isso centraliza o texto do componente `TextView`.
- **Text**: `@string/taggedSearches` – Crie um recurso de string com o valor "Tagged Searches".
- **Padding Top** (nó **View**): `@dimen/activity_vertical_margin` – Isso garante que o componente superior dentro do elemento `LinearLayout` vertical seja inserido com uma distância de 16dp a partir da margem superior do layout.

Com o componente `ListView` selecionado, configure as seguintes propriedades:

- **Width** (nó **Layout Parameters**): `match_parent`
- **Height** (nó **Layout Parameters**): `0dp` – O IDE recomenda esse valor quando a propriedade `Weight` também é configurada, para que possa organizar os componentes de forma mais eficiente.
- **Weight** (nó **Layout Parameters**): `1`
- **Gravity** (nó **Layout Parameters**): `fill` – O componente `ListView` deve preencher todo o espaço horizontal e vertical disponível.
- **Padding Top** (nó **View**): `@dimen/activity_vertical_margin` – Isso garante que o componente superior dentro do elemento `LinearLayout` vertical seja inserido com uma distância de 16dp a partir da margem superior do layout.
- **Top e Bottom** (localizadas no nó **Margins** do nó **Layout Parameters**): `@dimen/tagged_searches_padding` – Crie um novo recurso de dimensão `tagged_searches_padding` clicando no botão de reticências à direita da propriedade **Top**. Na caixa de diálogo **Resource Chooser**, clique em **New Dimension...** para criar um novo recurso de dimensão. Especifique `tagged_searches_padding` para **Name**, 8dp para **Value** e clique em

OK; então, selecione seu novo recurso de dimensão e clique em OK. Para a propriedade **Bottom**, basta selecionar esse novo recurso de dimensão. Essas propriedades garantem a existência de uma margem de 8dp entre o componente `TextView` e a parte superior do componente `ListView`, e entre a parte inferior de `ListView` e a parte inferior do componente `LinearLayout` vertical.

4.4.4 Barra de ferramentas do editor Graphical Layout

Agora você concluiu a interface gráfica do usuário de `MainActivity`. A barra de ferramentas do editor **Graphical Layout** (Fig. 4.14) contém vários botões que permitem visualizar o projeto para outros tamanhos e orientações de tela. Em particular, você pode ver imagens em miniatura de muitos tamanhos e orientações de tela clicando na seta que aponta para baixo ao lado do botão  e selecionando **Preview Representative Sample** ou **Preview All Screen Sizes**. Para cada miniatura existem botões + e - em que você pode clicar para ampliar e reduzir. A Figura 4.14 mostra alguns dos botões da barra de ferramentas do editor **Graphical Layout**.

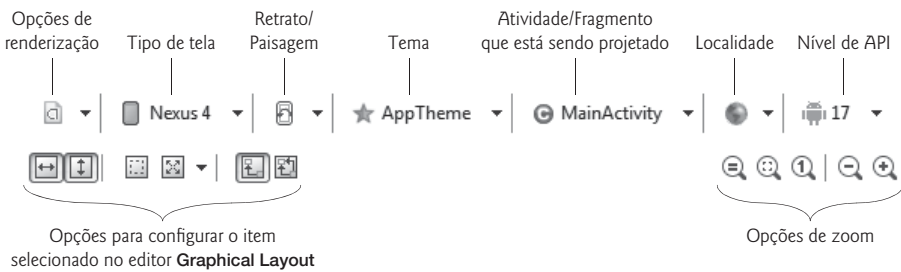


Figura 4.14 Opções de configuração da tela de desenho (canvas).

Opção	Descrição
Opções de renderização	Visualiza uma tela de projeto por vez ou mostra seu projeto em diversos tamanhos de tela simultaneamente.
Tipo de tela	O Android funciona em diversos dispositivos, de modo que o editor Graphical Layout vem com muitas configurações de dispositivo que representam vários tamanhos e resoluções de tela que podem ser usados para projetar sua interface gráfica do usuário. Neste livro, usamos as telas predefinidas do Nexus 4 , Nexus 7 e Nexus 10 , dependendo do aplicativo. Na Fig. 4.14, selecionamos Nexus 4 .
Retrato/Paisagem	Alterna a área de projeto entre as orientações <i>retrato</i> e <i>paisagem</i> .
Tema	Pode ser usado para definir o tema da interface gráfica do usuário.
Atividade/Fragmento que está sendo projetado	Mostra a classe Activity ou Fragment correspondente à interface gráfica do usuário que está sendo projetada.
Localidade	Para aplicativos <i>internacionalizados</i> (Seção 2.8), permite selecionar um local específico para que você possa ver, por exemplo, como seu projeto fica com strings em um idioma diferente.
Nível de API	Especifica o nível de API utilizado para o projeto. Com cada novo nível de API normalmente existem novos recursos de interface gráfica do usuário. A janela do editor Graphical Layout mostra somente os recursos que estão disponíveis no nível de API selecionado.

Figura 4.15 Explicação das opções de configuração da tela de desenho.

4.4.5 Layout do item ListView: list_item.xml

Ao preencher um componente ListView com dados, você deve especificar o formato aplicado a cada item da lista. Neste aplicativo, cada item exibe o nome de identificador de String de uma pesquisa salva. Para especificar a formatação de cada item da lista, você vai criar um novo layout contendo apenas um componente TextView com a formatação apropriada. Execute os passos a seguir:

1. Na janela **Package Explorer**, expanda a pasta **res** do projeto e, em seguida, clique com o botão direito do mouse na pasta **layout** e selecione **New > Other...** para exibir a caixa de diálogo **New**.
2. No nó **Android**, selecione **Android XML Layout File** e clique em **Next >** para exibir a caixa de diálogo da Fig. 4.16; em seguida, configure o arquivo como mostrado. O nome de arquivo do novo layout é **list_item.xml** e o elemento raiz do layout é um **TextView**.
3. Clique em **Finish** para criar o arquivo.

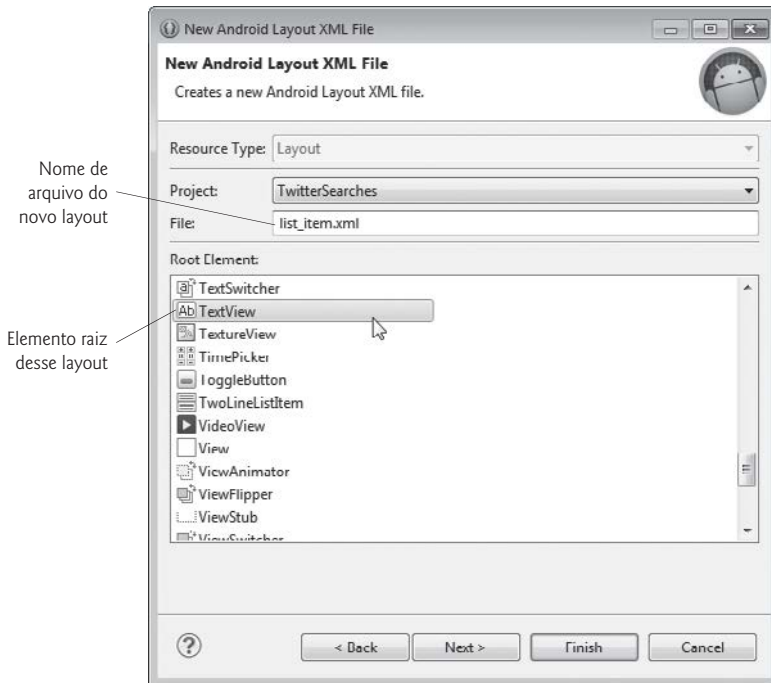


Figura 4.16 Criando um novo layout **list_item.xml** na caixa de diálogo **New Android Layout XML File**.

O IDE abre o novo layout no editor **Graphical Layout**. Selecione o componente **TextView** na janela **Outline** e, então, configure as seguintes propriedades:

- **Id: @+id/textView** – As propriedades **Id** dos componentes da interface gráfica do usuário começam com a primeira letra minúscula por convenção.
- **Height (nó Layout Parameters): ?android:attr/listPreferredItemHeight** – Esse valor é um recurso predefinido do Android que representa a altura preferida de um

item de lista para responder corretamente aos toques do usuário com chance mínima de tocar no item errado.



Observação sobre aparência e comportamento 4.4

As diretrizes de projeto do Android especificam que o tamanho mínimo recomendado para um item que pode ser tocado na tela seja de 48dp por 48dp. Para obter mais informações sobre dimensionamento e espaçamento na interface gráfica do usuário, consulte <http://developer.android.com/design/style/metrics-grids.html>.

- **Gravity** (nó **Layout Parameters**): `center_vertical` – O componente `TextView` deve ser centralizado verticalmente dentro do item de `ListView`.
- **Text Appearance** (nó **TextView**): `?android:attr/textAppearanceMedium` – Esse é o recurso de tema predefinido que especifica a medida da fonte para texto de tamanho médio.

Itens de lista que exibem vários tipos de dados

Se um item de lista vai exibir vários tipos de dados, você precisará de um layout que seja composto de vários elementos, e cada elemento precisará de um atributo `android:id`.

Outros recursos predefinidos do Android

Existem muitos recursos predefinidos do Android, como os utilizados para configurar as propriedades `Height` e `Text Appearance` de um item de lista. Veja uma lista completa em:

<http://developer.android.com/reference/android/R.attr.html>

Para usar um valor em seus layouts, especifique-o no formato

`?android:attr/nomeDoRecurso`

4.5 Construção da classe `MainActivity`

As Figuras 4.17 a 4.27 implementam a lógica do aplicativo **Twitter Searches** na classe `MainActivity`, a qual estende `ListActivity`. O código padrão da classe `MainActivity` incluiu um método `onCreateOptionsMenu`, o qual removemos porque não é usado neste aplicativo – vamos discutir `onCreateOptionsMenu` no Capítulo 5. Por toda esta seção, supomos que você criará os recursos de `String` necessários à medida que os encontrar nas descrições de código.

4.5.1 As instruções `package` e `import`

A Figura 4.17 mostra as instruções `package` e `import` do aplicativo. A instrução `package` (inserida na linha 4 pelo IDE quando você criou o projeto) indica que a classe nesse arquivo faz parte do pacote `com.deitel.twittersearches`. As linhas 6 a 26 importam as classes e interfaces utilizadas pelo aplicativo.

```
1 // MainActivity.java
2 // Gerencia suas pesquisas favoritas no Twitter para
3 // facilitar o acesso e exibir no navegador Web do dispositivo
4 package com.deitel.twittersearches;
5
6 import java.util.ArrayList;
```

Figura 4.17 Instruções `package` e `import` de `MainActivity`.

```

7  import java.util.Collections;
8
9  import android.app.AlertDialog;
10 import android.app.ListActivity;
11 import android.content.Context;
12 import android.content.DialogInterface;
13 import android.content.Intent;
14 import android.content.SharedPreferences;
15 import android.net.Uri;
16 import android.os.Bundle;
17 import android.view.View;
18 import android.view.View.OnClickListener;
19 import android.view.inputmethod.InputMethodManager;
20 import android.widget.AdapterView;
21 import android.widget.AdapterView.OnItemClickListener;
22 import android.widget.AdapterView.OnItemLongClickListener;
23 import android.widget.ArrayAdapter;
24 import android.widget.EditText;
25 import android.widget.ImageButton;
26 import android.widget.TextView;
27

```

Figura 4.17 Instruções package e import de MainActivity.

As linhas 6 e 7 importam as classes `ArrayList` e `Collections` do pacote `java.util`. Usamos a classe `ArrayList` para manter a lista de identificadores das pesquisas salvas e a classe `Collections` para *ordená-los* a fim de que apareçam em ordem alfabética. Das instruções `import` restantes, consideramos somente as dos recursos apresentados neste capítulo.

- A classe `AlertDialog` do pacote `android.app` (linha 9) é usada para exibir caixas de diálogo.
- A classe `ListActivity` do pacote `android.app` (linha 10) é a superclasse de `MainActivity`, a qual fornece o componente `ListView` do aplicativo e os métodos para manipulá-lo.
- A classe `Context` do pacote `android.content` (linha 11) dá acesso às informações sobre o ambiente no qual o aplicativo está sendo executado e permite usar vários serviços do Android. Vamos utilizar uma constante dessa classe quando ocultarmos o teclado virtual via programação, depois que o usuário salvar uma pesquisa.
- A classe `DialogInterface` do pacote `android.content` (linha 12) contém a interface aninhada `OnClickListener`. Implementamos essa interface para tratar os eventos que ocorrem quando o usuário toca em um botão em um componente `AlertDialog`.
- A classe `Intent` do pacote `android.content` (linha 13) é usada para criar um objeto que especifica uma *ação* a ser executada e os *dados* utilizados nessa ação – o Android utiliza objetos `Intent` para ativar as atividades apropriadas. Vamos usar essa classe para ativar o navegador Web do dispositivo, para exibir os resultados da busca no Twitter e para exibir um *selecionador de intenção* a fim de que o usuário possa escolher como vai compartilhar uma pesquisa.
- A classe `SharedPreferences` do pacote `android.content` (linha 14) é usada para manipular *pares chave-valor persistentes* que são armazenados em arquivos associados ao aplicativo.
- A classe `Uri` do pacote `android.net` (linha 15) permite converter uma URL para o formato exibido por um elemento `Intent` que ativa o navegador Web do dispositivo.

- A classe **View** do pacote **android.view** (linha 17) é utilizada em vários métodos de tratamento de eventos para representar o componente da interface gráfica com que o usuário interage para iniciar um evento.
- A classe **View** contém o componente **OnClickListener** aninhado da interface (linha 18). Implementamos essa interface para tratar o evento disparado quando o usuário toca no componente **ImageButton** para salvar uma pesquisa.
- A classe **InputMethodManager** do pacote **android.view.inputmethod** (linha 19) permite ocultar o teclado virtual quando o usuário salva uma pesquisa.
- O pacote **android.widget** (linhas 20 a 26) contém os componentes da interface gráfica do usuário e layouts utilizados nas interfaces gráficas do Android. A classe **AdapterView** (linha 20) é a classe base de **ListView** e é utilizada ao configurar o adaptador de **ListView** (o qual fornece os itens de **ListView**). A interface **AdapterView.OnItemClickListener** (linha 21) é implementada para responder quando o usuário *toca* em um item em um componente **ListView**. A interface **AdapterView.OnItemLongClickListener** (linha 22) é implementada para responder quando o usuário faz um *pressionamento longo* em um item em um componente **ListView**. A classe **ArrayAdapter** (linha 23) é usada para *vincular* itens a um componente **ListView**. A classe **ImageButton** (linha 25) representa um botão que exibe uma imagem.

4.5.2 Extensão de **ListActivity**

MainActivity (Figs. 4.18 a 4.27) é a única classe **Activity** do aplicativo **Twitter Searches**. Quando você criou o projeto **TwitterSearches**, o IDE gerou **MainActivity** como uma subclasse de **Activity** e forneceu a “casca” de um método sobrescrito **onCreate**, o qual toda subclasse de **Activity** *deve* sobrescrever. Mudamos a superclasse para **ListActivity** (Fig. 4.18, linha 28). Quando essa alteração é feita, o IDE não reconhece a classe **ListActivity**, de modo que é preciso atualizar suas instruções **import**. No IDE, você pode usar **Source > Organize Imports** para isso. O Eclipse sublinha todo nome de classe ou interface que não reconhece. Nesse caso, se você deixar o mouse uns instantes sobre o nome da classe ou interface, aparecerá uma lista de *correções rápidas*. Se o IDE reconhecer o nome, irá sugerir a instrução **import** ausente que precisa ser adicionada – basta clicar no nome para adicioná-la.

```
28 public class MainActivity extends ListActivity
29 {
```

Figura 4.18 A classe **MainActivity** é uma subclasse de **ListActivity**.

4.5.3 Campos da classe **MainActivity**

A Figura 4.19 contém as variáveis estáticas e de instância da classe **MainActivity**. A constante **String SEARCHES** (linha 31) representa o nome do arquivo que vai armazenar as pesquisas no dispositivo. As linhas 33 e 34 declaram os componentes **EditText** que vamos usar para acessar as consultas e identificadores digitados pelo usuário. A linha 35 declara a variável de instância **savedSearches** de **SharedPreferences**, a qual será usada para manipular os *pares chave-valor* que representam as pesquisas salvas do usuário. A linha 36 declara o elemento **ArrayList<String>** que vai armazenar os nomes de identificador ordenados das pesquisas do usuário. A linha 37 declara o elemento **ArrayAdapter<String>** que utiliza o conteúdo de **ArrayList<String>** como origem dos itens exibidos no componente **ListView** de **MainActivity**.

**Boa prática de programação 4.1**

Para ter maior legibilidade e capacidade de modificação, use constantes String para representar nomes de arquivo (e outras literais String) que não precisem ser adaptados a outros idiomas e, portanto, não sejam definidos em strings.xml.

```

30 // nome do arquivo XML de SharedPreferences que armazena as pesquisas salvas
31 private static final String SEARCHES = "searches";
32
33 private EditText queryEditText; // EditText onde o usuário digita uma consulta
34 private EditText tagEditText; // EditText onde o usuário identifica uma consulta
35 private SharedPreferences savedSearches; // pesquisas favoritas do usuário
36 private ArrayList<String> tags; // lista de identificadores das pesquisas salvas
37 private ArrayAdapter<String> adapter; // vincula identificadores a ListView
38

```

Figura 4.19 Campos da classe MainActivity.

4.5.4 Sobrescrevendo o método onCreate de Activity

O método onCreate (Fig. 4.20) é chamado pelo sistema:

- quando o aplicativo *é carregado*;
- se o processo do aplicativo tiver sido *eliminado* pelo sistema operacional enquanto o aplicativo estava em segundo plano e, então, é *restaurado*;
- sempre que a configuração é modificada, como quando o usuário *gira* o dispositivo ou quando *abre* ou *fecha* um teclado físico.

O método inicializa as variáveis de instância de Activity e componentes da interface gráfica do usuário – o mantemos simples para que o aplicativo seja carregado rapidamente. A linha 43 realiza a chamada *exigida* para o método onCreate da superclasse. Como no aplicativo anterior, a chamada para setContentView (linha 44) passa a constante R.layout.activity_main para *inflar a interface gráfica do usuário* de activity_main.xml.

```

39 // chamado quando MainActivity é criada
40 @Override
41 protected void onCreate(Bundle savedInstanceState)
42 {
43     super.onCreate(savedInstanceState);
44     setContentView(R.layout.activity_main);
45
46     // obtém referências para os EditText
47     queryEditText = (EditText) findViewById(R.id.queryEditText);
48     tagEditText = (EditText) findViewById(R.id.tagEditText);
49
50     // obtém os SharedPreferences que contêm as pesquisas salvas do usuário
51     savedSearches = getSharedPreferences(SEARCHES, MODE_PRIVATE);
52
53     // armazena os identificadores salvos em um ArrayList e, então, os ordena
54     tags = new ArrayList<String>(savedSearches.getAll().keySet());
55     Collections.sort(tags, String.CASE_INSENSITIVE_ORDER);
56
57     // cria ArrayAdapter e o utiliza para vincular os identificadores a ListView
58     adapter = new ArrayAdapter<String>(this, R.layout.list_item, tags);
59     setListAdapter(adapter);

```

Figura 4.20 Sobrescrevendo o método onCreate de Activity. (continua)


```

60
61 // registra receptor para salvar uma pesquisa nova ou editada
62 ImageButton saveButton =
63     (ImageButton) findViewById(R.id.saveButton);
64 saveButton.setOnClickListener(saveButtonListener);
65
66 // registra receptor que pesquisa quando o usuário toca em um identificador
67 listView.setOnItemClickListener(itemClickListener);
68
69 // configura o receptor que permite ao usuário excluir ou editar uma pesquisa
70 listView.setOnItemClickListener(itemLongClickListener);
71 } // fim do método onCreate
72

```

Figura 4.20 Sobrescrevendo o método onCreate de Activity.

Obtendo referências para os componentes *EditText*

As linhas 47 e 48 obtêm referências para os componentes *queryEditText* e *tagEditText* para inicializar as variáveis de instância correspondentes.

Obtendo um objeto *SharedPreferences*

A linha 51 usa o método **getSharedPreferences** (herdado da classe *Context*) para obter um objeto *SharedPreferences* que pode ler *pares identificador-consulta* (se houver) armazenados anteriormente no arquivo *SEARCHES*. O primeiro argumento indica o nome do arquivo que contém os dados. O segundo argumento especifica a acessibilidade do arquivo e pode ser configurado com uma das opções a seguir:

- **MODE_PRIVATE** – O arquivo é acessível *somente* para esse aplicativo. Na maioria dos casos, você vai usar esta opção.
- **MODE_WORLD_READABLE** – Qualquer aplicativo no dispositivo pode *ler* o arquivo.
- **MODE_WORLD_WRITABLE** – Qualquer aplicativo no dispositivo pode *gravar* no arquivo.

Essas constantes podem ser combinadas com o operador OU bit a bit (**|**). Não estamos lendo muitos dados neste aplicativo; portanto, ele é rápido o suficiente para carregar as pesquisas em *onCreate*.



Dica de desempenho 4.1

O acesso a dados extensos não deve ser feito na thread UI; caso contrário, o aplicativo exibirá uma caixa de diálogo Application Not Responding (ANR) – normalmente após cinco segundos de inatividade. Para obter informações sobre projeto de aplicativos ágeis nas respostas, consulte <http://developer.android.com/guide/practices/design/responsiveness.html>.

Obtendo as chaves armazenadas no objeto *SharedPreferences*

Queremos exibir os identificadores de pesquisa em ordem alfabética para que o usuário possa localizar facilmente uma pesquisa a ser feita. Primeiramente, a linha 54 obtém os objetos *String* que representam as chaves no objeto *SharedPreferences* e os armazena em identificadores (um elemento *ArrayList<String>*). O método **getAll** de *SharedPreferences* retorna todas as pesquisas salvas como um objeto *Map* (pacote *java.util*) – uma coleção de pares chave-valor. Então, chamamos o método **keySet** nesse objeto para obter todas as chaves como um objeto *Set* (pacote *java.util*) – uma coleção de valores únicos. O resultado é usado para inicializar *tags*.

Classificando o componente *ArrayList* de identificadores

A linha 55 usa `Collections.sort` para ordenar tags. Como o usuário pode inserir identificadores usando misturas de letras maiúsculas e minúsculas, optamos por fazer uma *ordenação sem diferenciação de maiúsculas e minúsculas*, passando o objeto predefinido `String.CASE_INSENSITIVE_ORDER` de `Comparator<String>` como o segundo argumento para `Collections.sort`.

Usando um componente *ArrayAdapter* para preencher o elemento *ListView*

Para exibir os resultados em um elemento `ListView`, criamos um novo objeto `ArrayAdapter<String>` (linha 58), o qual faz o mapeamento dos identificadores de conteúdo para os componentes `TextView` que são exibidos no elemento `ListView` de `MainActivity`. O construtor de `ArrayAdapter<String>` recebe:

- o objeto `Context` (`this`) no qual o elemento `ListView` é exibido – `this` é o componente `MainActivity`;
- o identificador do recurso (`R.layout.list_item`) do layout utilizado para exibir cada item no componente `ListView`;
- um objeto `List<String>` contendo os itens a exibir – `tags` é um objeto `ArrayList<String>`, o qual implementa a interface `List<String>`; portanto, `tags` é um objeto `List<String>`.

A linha 59 usa o método herdado `setListAdapter` de `ListActivity` para vincular o componente `ListView` ao elemento `ArrayAdapter`, para que `ListView` possa exibir os dados.

Registrando receptores para *saveButton* e *ListView*

As linhas 62 e 63 obtêm uma referência para o componente `saveButton`, e a linha 64 registra seu receptor – a variável de instância `saveButtonListener` faz referência a um *objeto de classe interna anônima* que implementa a interface `OnClickListener` (Fig. 4.21). A linha 67 usa o método herdado `getListView` de `ListActivity` para obter uma referência para o componente `ListView` dessa atividade e, então, registra `OnItemClickListener` de `ListView` – a variável de instância `itemClickListener` faz referência a um *objeto de classe interna anônima* que implementa essa interface (Fig. 4.24). Do mesmo modo, a linha 70 registra `OnItemLongClickListener` de `ListView` – a variável de instância `itemLongClickListener` faz referência a um *objeto de classe interna anônima* que implementa essa interface (Fig. 4.25).

4.5.5 Classe interna anônima que implementa a interface *OnClickListener* de *saveButton* para salvar uma pesquisa nova ou atualizada

A Figura 4.21 declara e inicializa a variável de instância `saveButtonListener`, a qual faz referência a um *objeto de classe interna anônima* que implementa a interface `OnClickListener`. A linha 64 (Fig. 4.20) registrou `saveButtonListener` como rotina de tratamento de eventos de `saveButton`. As linhas 76 a 109 sobrescrevem o método `onClick` da interface `OnClickListener`. Se o usuário digitou uma consulta e um identificador (linhas 80 e 81), as linhas 83 e 84 chamam o método `addTaggedSearch` (Fig. 4.23) para armazenar o par identificador-consulta e as linhas 85 e 86 limpam os dois componentes `EditText`. As linhas 88 a 90 ocultam o teclado virtual.

```

73 // saveButtonListener salva um par identificador-consulta em SharedPreferences
74 public OnClickListener saveButtonListener = new OnClickListener()
75 {
76     @Override
77     public void onClick(View v)
78     {
79         // cria identificador se nem queryEditText nem tagEditText está vazio
80         if (queryEditText.getText().length() > 0 &&
81             tagEditText.getText().length() > 0)
82         {
83             addTaggedSearch(queryEditText.getText().toString(),
84                             tagEditText.getText().toString());
85             queryEditText.setText(""); // limpa queryEditText
86             tagEditText.setText(""); // limpa tagEditText
87
88             ((InputMethodManager) getSystemService(
89                 Context.INPUT_METHOD_SERVICE)).hideSoftInputFromWindow(
90                 tagEditText.getWindowToken(), 0);
91         }
92     else // exibe mensagem solicitando que forneça uma consulta e um identificador
93     {
94         // cria um novo AlertDialog Builder
95         AlertDialog.Builder builder =
96             new AlertDialog.Builder(MainActivity.this);
97
98         // configura o título da caixa de diálogo e a mensagem a ser exibida
99         builder.setMessage(R.string.missingMessage);
100
101         // fornece um botão OK que simplesmente remove a caixa de diálogo
102         builder.setPositiveButton(R.string.OK, null);
103
104         // cria AlertDialog a partir de AlertDialog.Builder
105         AlertDialog errorDialog = builder.create();
106         errorDialog.show(); // exibe a caixa de diálogo modal
107     }
108 } // fim do método onClick
109 }; // fim da classe interna anônima OnClickListener
110

```

Figura 4.21 Classe interna anônima que implementa a interface `OnClickListener` de `saveButton` para salvar uma pesquisa nova ou atualizada.

Configurando um componente `AlertDialog`

Se o usuário não digitou uma consulta e um identificador, as linhas 92 a 108 exibem um elemento `AlertDialog` indicando que ele deve digitar ambos. Um objeto `AlertDialog.Builder` (linhas 95 e 96) ajuda a configurar e criar um componente `AlertDialog`. O argumento do construtor é o objeto `Context` no qual a caixa de diálogo vai ser exibida – neste caso, o elemento `MainActivity`, ao qual nos referimos por meio de sua referência `this`. Para acessar `this` a partir de uma *classe interna anônima*, você deve qualificá-lo totalmente com o nome da classe externa. A linha 99 configura a mensagem da caixa de diálogo com o recurso `String R.string.missingMessage` ("Enter both a Twitter search query and a tag").



Observação sobre aparência e comportamento 4.5

É possível configurar o título de `AlertDialog` (o qual aparece acima da mensagem da caixa de diálogo) com o método `setTitle` de `AlertDialog.Builder`. De acordo com as diretrizes de projeto do Android para caixas de diálogo (<http://developer.android.com/design/building-blocks/dialogs.html>), a maioria delas não precisa de títulos. Uma caixa de diálogo deve exibir um título para “uma operação de alto risco, envolvendo possível perda de dados, conectividade, cobranças extras e assim por diante”. Além disso, as caixas de diálogo que exibem listas de opções utilizam o título para especificar sua finalidade.

Adicionando recursos de *String* a *strings.xml*

Para criar recursos de *String*, como `R.string.missingMessage`, abra o arquivo *strings.xml*, localizado na pasta *res/values* do projeto. O IDE mostra esse arquivo em um *editor de recursos* que tem duas guias – *Resources* e *strings.xml*. Na caixa de diálogo *Resources*, clique em *Add...* para exibir a caixa de diálogo da Fig. 4.22. Selecionar *String* e clicar em *OK* exibe os campos de texto *Name* e *Value*, em que você pode digitar um novo nome (por exemplo, `missingMessage`) e um novo valor de recurso *String*. Salve seu arquivo *strings.xml* após fazer as alterações. Você também pode usar a guia *Resource* do editor de recursos para selecionar um recurso *String* já existente a fim de alterar seu nome e valor.

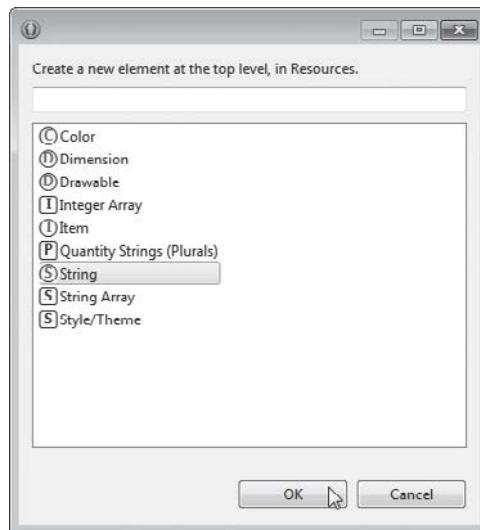


Figura 4.22 Adicionando um recurso de *String*.

Especificando o botão positivo do componente *AlertDialog*

Neste componente *AlertDialog*, precisamos de apenas um botão que permita ao usuário aceitar a mensagem. Especificamos isso como o botão *positivo* da caixa de diálogo (Fig. 4.21, linha 102) – tocar nesse botão indica que o usuário aceita a mensagem exibida na caixa de diálogo. O método `setPositiveButton` recebe o rótulo do botão (especificado com o recurso de *String* `R.string.OK`) e uma referência para a rotina de tratamento de eventos do botão. Para essa caixa de diálogo não precisamos responder ao evento; portanto, especificamos `null` para a rotina de tratamento de eventos. Quando o usuário toca no botão, a caixa de diálogo simplesmente desaparece da tela.

Criação e exibição do componente *AlertDialog*

Você cria o componente *AlertDialog* chamando o método `create` de *AlertDialog.Builder* (linha 105) e exibe a *caixa de diálogo modal* chamando o método `show` de *AlertDialog* (linha 106).

4.5.6 Método `addTaggedSearch`

A rotina de tratamento de eventos da Fig. 4.21 chama o método `addTaggedSearch` de *MainActivity* (Fig. 4.23) para adicionar uma nova pesquisa a *savedSearches* ou para modificar uma pesquisa já existente.

```

111 // adiciona uma nova pesquisa ao arquivo de salvamento e, então, atualiza todos
112 // os componentes Button
113 private void addTaggedSearch(String query, String tag)
114 {
115     // obtém um objeto SharedPreferences.Editor para armazenar novos pares
116     // identificador-consulta
117     SharedPreferences.Editor preferencesEditor = savedSearches.edit();
118     preferencesEditor.putString(tag, query); // armazena pesquisa atual
119     preferencesEditor.apply(); // armazena as preferências atualizadas
120
121 // se o identificador é novo, adiciona-o, ordena tags e exibe a lista atualizada
122 if (!tags.contains(tag))
123 {
124     tags.add(tag); // adiciona o novo identificador
125     Collections.sort(tags, String.CASE_INSENSITIVE_ORDER);
126     adapter.notifyDataSetChanged(); // vincula os identificadores a ListView
127 }
128 }

```

Figura 4.23 Método addTaggedSearch da classe MainActivity.

Editando o conteúdo de um objeto SharedPreferences

Para alterar o conteúdo de um objeto `SharedPreferences`, você deve primeiramente chamar seu método `edit` a fim de obter um objeto `SharedPreferences.Editor` (linha 115), o qual pode adicionar e remover pares chave-valor e modificar o valor associado a uma chave específica em um arquivo `SharedPreferences`. A linha 116 chama o método `putString` de `SharedPreferences.Editor` para salvar o identificador da pesquisa (a chave) e a consulta (o valor correspondente) – caso o identificador já exista no objeto `SharedPreferences`, isso atualiza o valor. A linha 117 efetiva as alterações, chamando o método `apply` de `SharedPreferences.Editor` para fazer as mudanças no arquivo.

Notificando o componente ArrayAdapter de que seus dados mudaram

Quando o usuário adiciona uma nova pesquisa, o componente `ListView` deve ser atualizado para exibi-la. As linhas 120 a 125 determinam se um novo identificador foi adicionado. Em caso positivo, as linhas 122 e 123 adicionam o identificador da nova pesquisa a tags e, então, ordenam a coleção. A linha 124 chama o método `notifyDataSetChanged` de `ArrayAdapter` para indicar que os dados subjacentes em tags foram alterados. Então, o adaptador notifica o componente `ListView` para que atualize sua lista de itens exibidos.

4.5.7 Classe interna anônima que implementa a interface `OnItemClickListener` de `ListView` para exibir resultados de pesquisa

A Figura 4.24 declara e inicializa a variável de instância `itemClickListener`, a qual faz referência a um *objeto de classe interna anônima* que implementa a interface `OnItemClickListener`. A linha 67 (Fig. 4.20) registrou `itemClickListener` como a rotina de tratamento de eventos de `ListView` que responde quando o usuário *toca* em um item no componente `ListView`. As linhas 131 a 145 sobrescrevem o método `onItemClick` da interface `OnItemClickListener`. Os argumentos do método são:

- O elemento `AdapterView` em que o usuário tocou em um item. O `?` em `AdapterView<?>` é um *curinga* nos tipos genéricos Java indicando que o método `onItemClick` pode receber um objeto `AdapterView` que exibe *qualquer* tipo de dado – neste caso, `ListView<String>`.

- O elemento View em que o usuário tocou no objeto AdapterView – neste caso, o componente TextView que exibe o identificador de uma busca.
- O número do índice, com *base zero*, do item em que o usuário tocou.
- O identificador da linha do item que foi tocado – isso é usado principalmente para dados obtidos de um banco de dados (como você vai fazer no Capítulo 8).

```

128 // itemClickListener ativa o navegador Web para exibir resultados da busca
129 OnItemClickListener itemClickListener = new OnItemClickListener()
130 {
131     @Override
132     public void onItemClick(AdapterView<?> parent, View view,
133         int position, long id)
134     {
135         // obtém a string de consulta e cria uma URL representando a busca
136         String tag = ((TextView) view).getText().toString();
137         String urlString = getString(R.string.searchURL) +
138             Uri.encode(savedSearches.getString(tag, ""), "UTF-8");
139
140         // cria um objeto Intent para ativar um navegador Web
141         Intent webIntent = new Intent(Intent.ACTION_VIEW,
142             Uri.parse(urlString));
143
144         startActivity(webIntent); // ativa o navegador Web para ver os resultados
145     }
146 }; // fim da declaração de itemClickListener
147

```

Figura 4.24 Classe interna anônima que implementa a interface `OnItemClickListener` de `ListView` para exibir os resultados da pesquisa.

Obtendo recursos de *String*

A linha 136 obtém o texto do elemento View em que o usuário tocou no componente `ListView`. As linhas 137 e 138 criam um objeto `String` contendo a URL de busca no Twitter e a consulta a ser feita. Primeiramente, a linha 137 chama o método `getString` herdado de `Activity` com apenas um argumento para obter o recurso `String` chamado `searchURL`, o qual contém a URL da página de busca do Twitter:

```
http://mobile.twitter.com/search/
```

Como em todos os recursos de `String` neste aplicativo, você deve adicionar esse recurso a `strings.xml`.

Obtendo objetos *String* a partir de um objeto *SharedPreferences*

Anexamos o resultado da linha 138 à URL de busca para completar o objeto `urlString`. O método `getString` de `SharedPreferences` retorna a consulta associada ao identificador. Se o identificador ainda não existe, o segundo argumento (" ", neste caso) é retornado. A linha 138 passa a consulta para o método `encode` de `Uri`, o qual *faz o escape* de quaisquer caracteres de URL especiais (como `?`, `/`, `:`, etc.) e retorna o assim chamado objeto `String` *codificado em URL*. Isso é importante para garantir que o servidor Web do Twitter que recebe o pedido possa analisar a URL adequadamente para obter a consulta de busca.

Criando um objeto *Intent* para ativar o navegador Web do dispositivo

As linhas 141 e 142 criam um novo objeto `Intent`, o qual vamos usar para *ativar o navegador Web* do dispositivo e exibir os resultados da busca. Os objetos `Intent` podem ser usados para ativar outras atividades no mesmo aplicativo ou em outros aplicativos. O primeiro argumento do construtor do objeto `Intent` é uma constante descrevendo a

ação a ser executada. `Intent.ACTION_VIEW` indica que queremos exibir uma representação dos dados. Na classe `Intent` são definidas muitas constantes que descrevem ações como *pesquisar*, *escolher*, *enviar* e *reproduzir*. O segundo argumento (linha 142) é um `Uri` (uniform resource identifier) representando os *dados* nos quais queremos executar a ação. O método `parse` da classe `Uri` converte um objeto `String` representando uma URL (uniform resource locator) em um `Uri`.

Iniciando uma atividade para um objeto `Intent`

A linha 144 passa o objeto `Intent` para o método herdado `startActivity` de `Activity`, o qual inicia uma atividade que pode executar a *ação* especificada nos *dados* fornecidos. Neste caso, como especificamos a visualização de um URI, o objeto `Intent` ativa o navegador Web do dispositivo para exibir a página Web correspondente. Essa página mostra os resultados da busca fornecida no Twitter.

4.5.8 Classe interna anônima que implementa a interface `OnItemLongClickListener` de `ListView` para compartilhar, editar ou excluir uma pesquisa

A Figura 4.25 declara e inicializa a variável de instância `itemLongClickListener`, a qual faz referência a um *objeto de classe interna anônima* que implementa a interface `OnItemLongClickListener`. A linha 70 (Fig. 4.20) registrou `itemLongClickListener` como a rotina de tratamento de eventos de `ListView` que responde quando o usuário faz um pressionamento longo em um item no componente `ListView`. As linhas 153 a 210 sobrescrevem o método `onItemLongClick` da interface `OnItemLongClickListener`.

```

148 // itemLongClickListener exibe uma caixa de diálogo que permite ao usuário excluir
149 // ou editar uma pesquisa salva
150 OnItemLongClickListener itemLongClickListener =
151     new OnItemLongClickListener()
152     {
153         @Override
154         public boolean onItemLongClick(AdapterView<?> parent, View view,
155             int position, long id)
156         {
157             // obtém o identificador em que o usuário fez o pressionamento longo
158             final String tag = ((TextView) view).getText().toString();
159
160             // cria um novo componente AlertDialog
161             AlertDialog.Builder builder =
162                 new AlertDialog.Builder(MainActivity.this);
163
164             // configura o título do componente AlertDialog
165             builder.setTitle(
166                 getString(R.string.shareEditDeleteTitle, tag));
167
168             // define a lista de itens a exibir na caixa de diálogo
169             builder.setItems(R.array.dialog_items,
170                 new DialogInterface.OnClickListener()
171                 {
172                     // responde ao toque do usuário, compartilhando, editando ou
173                     // excluindo uma pesquisa salva
174                     @Override
175                     public void onClick(DialogInterface dialog, int which)

```

Figura 4.25 Classe interna anônima que implementa a interface `OnItemLongClickListener` de `ListView` para compartilhar, editar ou excluir.

```

176         {
177             switch (which)
178             {
179                 case 0: // compartilha
180                     shareSearch(tag);
181                     break;
182                 case 1: // edita
183                     // configura componentes EditText para corresponder ao
184                     // identificador e à consulta escolhidos
185                     tagEditText.setText(tag);
186                     queryEditText.setText(
187                         savedSearches.getString(tag, ""));
188                     break;
189                 case 2: // exclui
190                     deleteSearch(tag);
191                     break;
192             }
193         } // fim de DialogInterface.OnClickListener
194     }; // fim da chamada a builder.setItems
195
196     // configura o componente Button negativo de AlertDialog
197     builder.setNegativeButton(getString(R.string.cancel),
198         new DialogInterface.OnClickListener()
199     {
200         // chamado quando o componente Button "Cancel" é clicado
201         public void onClick(DialogInterface dialog, int id)
202         {
203             dialog.cancel(); // remove o componente AlertDialog
204         }
205     }); // fim da chamada a setNegativeButton
206
207     builder.create().show(); // exibe o componente AlertDialog
208     return true;
209 } // fim do método onItemClick
210 }; // fim da declaração de onItemClick
211
212

```

Figura 4.25 Classe interna anônima que implementa a interface `OnItemLongClickListener` de `ListView` para compartilhar, editar ou excluir.

Variáveis locais `final` para uso em classes internas anônimas

A linha 158 obtém o texto do item em que o usuário fez o *pressionamento longo* e o atribui à variável local `final` chamada `tag`. Toda variável local ou parâmetro de método que vai ser usado em uma classe interna anônima *precisa* ter a declaração `final`.

Componente `AlertDialog` que exibe uma lista de itens

As linhas 161 a 166 criam um elemento `AlertDialog.Builder` e configuram o título da caixa de diálogo com um objeto `String` formatado, no qual `tag` substitui o especificador de formato no recurso `R.string.shareEditDeleteTitle` (o qual representa "Share, Edit or Delete the search tagged as \"%s\"). A linha 166 chama o método herdado `getString` de `Activity`, o qual recebe *vários argumentos* – o primeiro é o identificador de um recurso `String` representando um objeto *String de formato*, e os restantes são os valores que devem substituir os especificadores de formato no objeto `String` de formato. Além dos botões, um componente `AlertDialog` pode exibir uma lista de itens em um elemento `ListView`. As linhas 169 a 194 especificam que a caixa de diálogo deve exibir o array de objetos `String` `R.array.dialog_items` (os quais representam os objetos `String`

"Share", "Edit" e "Delete") e definem uma classe interna anônima para responder quando o usuário tocar em um item da lista.

Adicionando um recurso de array de Strings a strings.xml

O array de objetos String é definido como um recurso de array de Strings no arquivo strings.xml. Para adicionar um recurso de array de Strings a strings.xml:

1. Siga os passos da Seção 4.5.5 para adicionar um recurso String, mas selecione **String Array**, em vez de **String**, na caixa de diálogo da Fig. 4.22; em seguida, clique em **OK**.
2. Especifique o nome do array (dialog_items) no campo de texto **Name**.
3. Selecione o array na lista de recursos, no lado esquerdo do editor de recursos.
4. Clique em **Add...** e depois em **OK** para adicionar um novo objeto **Item** ao array.
5. Especifique o valor do novo objeto **Item** no campo de texto **Value**.

Execute esses passos para os itens Share, Edit e Delete (nessa ordem) e, então, salve o arquivo strings.xml.

Rotina de tratamento de eventos para a lista de itens da caixa de diálogo

A classe interna anônima nas linhas 170 a 193 determina o item selecionado pelo usuário na lista da caixa de diálogo e executa a ação apropriada. Se o usuário seleciona **Share**, o método shareSearch é chamado (linha 180). Se o usuário seleciona **Edit**, as linhas 184 a 186 exibem a consulta e o identificador da pesquisa (tag) nos componentes EditText. Se o usuário seleciona **Delete**, deleteSearch é chamado (linha 189).

Configurando o botão negativo e exibindo a caixa de diálogo

As linhas 197 a 206 configuram o botão *negativo* da caixa de diálogo para removê-la caso o usuário decida não compartilhar, editar ou excluir a pesquisa. A linha 208 cria e exibe a caixa de diálogo.

4.5.9 Método shareSearch

O método shareSearch (Fig. 4.26) é chamado pela rotina de tratamento de eventos da Fig. 4.25 quando o usuário opta por compartilhar uma pesquisa. As linhas 217 e 218 criam um objeto String representando a pesquisa a ser compartilhada. As linhas 221 a 227 criam e configuram um objeto Intent que permite ao usuário enviar a URL da busca usando uma atividade que pode manipular Intent.ACTION_SEND (linha 222).

```

213 // permite escolher um aplicativo para compartilhar a URL de uma pesquisa salva
214 private void shareSearch(String tag)
215 {
216     // cria a URL que representa a pesquisa
217     String urlString = getString(R.string.searchURL) +
218         Uri.encode(savedSearches.getString(tag, ""), "UTF-8");
219
220     // cria um objeto Intent para compartilhar urlString
221     Intent shareIntent = new Intent();
222     shareIntent.setAction(Intent.ACTION_SEND);
223     shareIntent.putExtra(Intent.EXTRA_SUBJECT,
224         getString(R.string.shareSubject));

```

Figura 4.26 Método shareSearch da classe MainActivity.

```

225     shareIntent.putExtra(Intent.EXTRA_TEXT,
226         getString(R.string.shareMessage, urlString));
227     shareIntent.setType("text/plain");
228
229     // exibe os aplicativos que podem compartilhar texto
230     startActivity(Intent.createChooser(shareIntent,
231         getString(R.string.shareSearch)));
232 }
233

```

Figura 4.26 Método `shareSearch` da classe `MainActivity`.

Adicionando extras a um objeto *Intent*

Um objeto `Intent` inclui um elemento `Bundle` de *extras* – informações adicionais passadas para a atividade que manipula o objeto `Intent`. Por exemplo, uma atividade de e-mail pode receber *extras* representando o assunto do e-mail, endereços CC e BCC, e o miolo do texto. As linhas 223 a 226 usam o método `putExtra` do objeto `Intent` para adicionar um extra como um par chave-valor ao elemento `Bundle` do `Intent`. O primeiro argumento do método é uma chave `String` representando a finalidade do extra e o segundo são os dados extras correspondentes. Os extras podem ser valores de tipo primitivo, arrays de tipo primitivo, objetos `Bundle` inteiros e muito mais – consulte a documentação da classe `Intent` para ver uma lista completa das sobrecargas de `putExtra`.

O extra nas linhas 223 e 224 especifica o assunto de um e-mail com o recurso `String R.string.shareSubject` ("Twitter search that might interest you"). Para uma atividade que *não* usa um assunto (como compartilhamento em uma rede social), esse extra é *ignorado*. O extra nas linhas 225 e 226 representa o texto a ser compartilhado – um objeto `String` formatado, no qual `urlString` é substituído no recurso `String R.string.shareMessage` ("Check out the results of this Twitter search: %s"). A linha 227 configura o tipo MIME do objeto `Intent` como `text/plain` – esses dados podem ser manipulados por uma atividade capaz de enviar mensagens de texto puro.

Exibindo um selecionador de intenção

Para exibir o *selecionador de intenção* mostrado na Fig. 4.8(a), passamos o objeto `Intent` e um título `String` para o método estático `createChooser` de `Intent` (linha 230). O recurso `R.string.shareSearch` ("Share Search to:") é usado como título do selecionador de intenção. É importante definir esse título para lembrar o usuário de selecionar uma atividade apropriada. Você não pode controlar os aplicativos instalados no telefone de um usuário nem os filtros de `Intent` que podem ativar esses aplicativos; portanto, é possível que atividades incompatíveis apareçam no selecionador. O método `createChooser` retorna um objeto `Intent`, que passamos para `startActivity` a fim de exibir o selecionador de intenção.

4.5.10 Método `deleteSearch`

A rotina de tratamento de eventos da Fig. 4.25 chama o método `deleteSearch` (Figura 4.27) quando o usuário faz um pressionamento longo no identificador de uma pesquisa e seleciona **Delete**. Antes de excluir a pesquisa, o aplicativo exibe um elemento `AlertDialog` para confirmar a operação de exclusão. As linhas 241 e 242 criam o título da caixa de diálogo com um objeto `String` formatado, no qual tag substitui o especificador de formato no recurso `R.string.confirmMessage` ("Are you sure you want to delete the search \"%s\"?"). As linhas 245 a 254 configuram o botão negativo da caixa de diálogo

para removê-la. O recurso `String R.string.cancel` representa "Cancel". As linhas 257 a 275 configuram o botão positivo da caixa de diálogo para remover a pesquisa. O recurso `String R.string.delete` representa "Delete". A linha 263 remove o identificador da coleção de identificadores, e as linhas 266 a 269 utilizam `SharedPreferences.Editor` para remover a busca do elemento `SharedPreferences` do aplicativo. Então, a linha 272 notifica o elemento `ArrayAdapter` de que os dados subjacentes mudaram para que o componente `ListView` possa atualizar sua lista de itens exibidos.

```

234 // exclui uma pesquisa depois que o usuário confirma a operação de exclusão
235 private void deleteSearch(final String tag)
236 {
237     // cria um novo componente AlertDialog
238     AlertDialog.Builder confirmBuilder = new AlertDialog.Builder(this);
239
240     // configura a mensagem do componente AlertDialog
241     confirmBuilder.setMessage(
242         getString(R.string.confirmMessage, tag));
243
244     // configura o componente Button negativo de AlertDialog
245     confirmBuilder.setNegativeButton(getString(R.string.cancel),
246         new DialogInterface.OnClickListener()
247         {
248             // chamado quando o componente Button "Cancel" é clicado
249             public void onClick(DialogInterface dialog, int id)
250             {
251                 dialog.cancel(); // remove a caixa de diálogo
252             }
253         }
254     ); // fim da chamada a setNegativeButton
255
256     // configura o componente Button positivo de AlertDialog
257     confirmBuilder.setPositiveButton(getString(R.string.delete),
258         new DialogInterface.OnClickListener()
259         {
260             // chamado quando o componente Button "Cancel" é clicado
261             public void onClick(DialogInterface dialog, int id)
262             {
263                 tags.remove(tag); // remove o identificador de tags
264
265                 // obtém o SharedPreferences.Editor para remover pesquisa salva
266                 SharedPreferences.Editor preferencesEditor =
267                     savedSearches.edit();
268                 preferencesEditor.remove(tag); // remove a pesquisa
269                 preferencesEditor.apply(); // salva as alterações
270
271                 // vincula novamente o ArrayList de identificadores a ListView para
272                 // mostrar a lista atualizada
273                 adapter.notifyDataSetChanged();
274             } // fim de OnClickListener
275         } // fim da chamada a setPositiveButton
276
277     confirmBuilder.create().show(); // exibe o componente AlertDialog
278 } // fim do método deleteSearch
279 } // fim da classe MainActivity

```

Figura 4.27 Método `deleteSearch` da classe `MainActivity`.

4.6 AndroidManifest.xml

Na Seção 3.6, você fez duas alterações no arquivo `AndroidManifest.xml`:

- A primeira indicou que o aplicativo **Tip Calculator** suportava somente a orientação retrato.
- A segunda forçava a exibição do teclado virtual quando o aplicativo começava a executar, para que o usuário pudesse inserir imediatamente o valor de uma conta no aplicativo **Tip Calculator**.

Este aplicativo aceita as orientações retrato e paisagem. Nenhuma alteração é necessária para indicar isso, pois todos os aplicativos aceitam as duas orientações por padrão.

A maioria dos usuários deste aplicativo vai ativá-lo para executar uma de suas pesquisas salvas. Quando o primeiro componente da interface gráfica do usuário é `EditText`, o Android dá o foco a ele quando o aplicativo é carregado. Como você sabe, quando um componente `EditText` recebe o foco, o teclado virtual correspondente é exibido (a não ser que um teclado físico esteja presente). Neste aplicativo, queremos impedir a exibição do teclado virtual, a não ser que o usuário toque em um dos componentes `EditText`. Para fazer isso, siga os passos da Seção 3.6 para configurar a opção **Window soft input mode**, mas defina seu valor como `stateAlwaysHidden`.

4.7 Para finalizar

Neste capítulo, você criou o aplicativo **Twitter Searches**. Primeiramente, você projetou a interface gráfica do usuário. Apresentamos o componente `ListView` para exibir uma lista de itens rolante e o utilizamos para exibir a grande lista de pesquisas salvas. Cada pesquisa foi associada a um item no componente `ListView`, em que o usuário podia tocar a fim de passar a pesquisa para o navegador Web do dispositivo. Você também aprendeu a criar recursos de `String` para usar em seu código Java.

Armazenamos os pares identificador-consulta de pesquisa em um arquivo `SharedPreferences` associado ao aplicativo e mostramos como *ocultar* o teclado virtual via programação. Também usamos um objeto `SharedPreferences.Editor` para armazenar, modificar e remover valores de um arquivo `SharedPreferences`. Em resposta ao toque do usuário em um identificador de pesquisa, carregamos um `Uri` no navegador Web do dispositivo, criando um novo objeto `Intent` e passando-o para o método `startActivity` de `Context`. Você também usou um objeto `Intent` para exibir um selecionador de intenção, permitindo ao usuário selecionar uma atividade para compartilhar uma busca.

Você usou objetos `AlertDialog.Builder` para configurar e criar componentes `AlertDialog` a fim de exibir mensagens para o usuário. Por fim, discutimos o arquivo `AndroidManifest.xml` e mostramos como configurar o aplicativo de modo que o teclado virtual não aparecesse quando o aplicativo fosse ativado.

No Capítulo 5, você vai construir o aplicativo **Flag Quiz**, no qual o usuário vê um elemento gráfico, que é a bandeira de um país, e precisa adivinhar qual é esse país em 3, 6 ou 9 tentativas. Você vai usar um menu e caixas de seleção para personalizar o teste, limitando as bandeiras e os países escolhidos a regiões específicas do mundo.

Exercícios de revisão

- 4.1** Preencha os espaços em branco em cada um dos seguintes enunciados:
- a) Os _____ normalmente são usados para ativar atividades – eles indicam uma ação a ser executada e os dados sobre os quais essa ação deve ser executada.
 - b) Implementamos a interface _____ para tratar os eventos que ocorrem quando o usuário toca em um botão em um componente `AlertDialog`.
 - c) O acesso a dados extensos não deve ser feito na thread UI; caso contrário, o aplicativo exibirá uma caixa de diálogo _____ – normalmente, após cinco segundos de inatividade.
 - d) Um objeto `Intent` é a descrição de uma ação a ser executada com _____ associados.
 - e) Os objetos `Intent` _____ especificam uma classe `Activity` exata a ser executada no mesmo aplicativo.
 - f) Quando você cria o projeto de cada aplicativo Android no Eclipse, o Plugin ADT gera e configura o arquivo _____ (também conhecido como manifesto do aplicativo), o qual descreve informações sobre o aplicativo.
 - g) Um _____ exibe uma lista de itens que pode ser rolada caso a lista completa não possa ser exibida na tela.
 - h) Um _____ cria um objeto que preenche o elemento `ListView` utilizando dados de um objeto coleção `ArrayList`.
 - i) _____ é uma subclasse de `ImageView`, a qual fornece recursos adicionais que permitem utilizar uma imagem, como um objeto `Button`.
- 4.2** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- a) Operações de entrada/saída extensas devem ser efetuadas na thread UI; caso contrário, isso afetará a rapidez de resposta de seu aplicativo.
 - b) Você chama `toArray` (com um array de `String` vazio como argumento) no objeto `Set` para converter o objeto `Set` em um array de `Strings`.
 - c) Quando a tarefa principal de uma atividade é exibir uma lista de itens rolante, você pode estender a classe `ListActivity`, a qual usa um componente `ListView` que ocupa o aplicativo inteiro como layout padrão.
 - d) `ListView` é uma subclasse de `Adapter` – um componente de interface gráfica do usuário é vinculado a uma fonte de dados.
 - e) Ao personalizar a interface gráfica do usuário de uma subclasse de `ListActivity`, o layout deve conter um componente `ListView` com o atributo `id` configurado como `"@android:id/list"` — o nome utilizado pela classe `ListActivity` para fazer referência a seu componente `ListView`.

Respostas dos exercícios de revisão

- 4.1** a) objetos `Intent`. b) `OnClickListener`. c) `Application Not Responding (ANR)`. d) dados. e) explícitos. f) `AndroidManifest.xml`. g) componente `ListView`. h) componente `ArrayAdapter`. i) `ImageButton`.
- 4.2** a) Falsa. As operações de entrada/saída extensas *não* devem ser efetuadas na thread UI, pois isso afetaria a rapidez de resposta de seu aplicativo. b) Verdadeira. c) Verdadeira. d) Falsa. `ListView` é uma subclasse de `AdapterView` – um componente de interface gráfica do usuário é vinculado a uma fonte de dados por meio de um objeto `Adapter`. e) Verdadeira.

Exercícios

- 4.3** Preencha os espaços em branco em cada um dos seguintes enunciados:
- Um layout preenche a área cliente inteira da tela se suas propriedades `Width` e `Height` (na seção **Layout Parameters** da janela **Properties**) são configuradas como _____.
 - O objeto _____ armazena pares chave-valor.
 - _____ (um método estático da classe `Collections` do pacote `java.util`) ordena o componente `List` em seu primeiro argumento.
 - Você usa objetos _____ para configurar e criar componentes `AlertDialog` a fim de exibir mensagens para o usuário.
 - O Android usa uma técnica conhecida como _____ para transmitir informações entre atividades dentro de um aplicativo ou atividades em aplicativos separados.
 - Um _____ é uma interface gráfica do usuário que mostra uma lista de aplicativos que podem manipular um objeto `Intent` especificado.
- 4.4** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- Um algoritmo especifica uma ação a ser realizada e os dados a serem afetados – o Android usa algoritmos para ativar as atividades apropriadas.
 - Você implementa a interface `View.OnClickListener` do pacote `android.view` para especificar o código que deve ser executado quando o usuário tocar em um componente `Button`.
 - O primeiro argumento passado para o construtor do objeto `Intent` são os dados a serem afetados.
 - Um objeto `Intent` explícito permite ao sistema iniciar a atividade mais adequada com base no tipo de dado.

Exercícios de projeto

- 4.5** (**Aplicativo Favorite Websites**) Usando as técnicas aprendidas neste capítulo, crie um aplicativo **Favorite Websites** que permita ao usuário gerar uma lista de sites favoritos.
- 4.6** (**Aprimoramentos no aplicativo Twitter Searches**) Aprimore o aplicativo **Twitter Searches** para permitir ao usuário adicionar filtros às pesquisas (por exemplo, incluir somente tweets com vídeos, imagens ou links). Talvez seja necessário investigar mais a fundo os operadores de pesquisa do Twitter.
- 4.7** (**Aplicativo Flickr Searches**) Investigue a API de pesquisa de fotos do Flickr (<http://www.flickr.com/services/api/flickr.photos.search.html>) e, então, implemente novamente o aplicativo **Twitter Searches** deste capítulo como um aplicativo **Flickr Searches**.
- 4.8** (**Aplicativo Flickr Searches aprimorado**) Aprimore o aplicativo **Flickr Searches** do Exercício 4.7 para permitir ao usuário adicionar filtros às pesquisas (por exemplo, incluir somente imagens contendo determinada cor, forma, objeto, etc.).
- 4.9** (**Jogo Word Scramble**) Crie um aplicativo que misture as letras de uma palavra ou frase e peça ao usuário para que digite a palavra ou frase correta. Monitore o placar mais alto do usuário no componente `SharedPreferences` do aplicativo. Inclua níveis (palavras com três, quatro, cinco, seis e sete letras). Como dica para o usuário, forneça uma definição com cada palavra. [Opcional: encontre um web service de dicionário gratuito e utilize-o para selecionar as palavras e definições.]

Exercícios de projeto avançados

4.10 (*Aplicativo Blackjack*) Crie um aplicativo de jogo de cartas Blackjack (Vinte-e-Um). São distribuídas duas cartas para o carteador e para o jogador. (Imagens de cartas são fornecidas com os exemplos do livro.) As cartas do jogador são distribuídas abertas. Apenas a primeira carta do carteador é distribuída aberta. Cada carta tem um valor. As cartas com numeração de 2 a 10 valem o seu valor de face. Valetes, damas e reis valem 10. Os ases podem valer 1 ou 11 — o que for mais vantajoso para o jogador. Se a soma das duas cartas iniciais do jogador for 21 (ou seja, o jogador recebeu uma carta de valor 10 e um ás, o qual valeria 11 nessa situação), ele tem um “vinte e um puro” e a carta fechada do carteador é revelada. Se o carteador não tiver um vinte e um puro, o jogador ganha o jogo imediatamente; caso contrário, a mão deu “empate” e ninguém ganha. Se o jogador não fez vinte e um com duas cartas, pode começar a receber mais cartas, uma por vez. Essas cartas são distribuídas abertas, e o jogador decide quando quer parar de receber cartas. Se o jogador “estourar” (isto é, a soma de suas cartas ultrapassar 21), o jogo termina e o jogador perde. Quando o jogador disser que quer parar de receber cartas, a carta oculta do carteador é revelada. Se o total do carteador for 16 ou menos, ele deve receber outra carta; caso contrário, deve parar de receber cartas. O carteador deve continuar a receber cartas até que a soma de suas cartas seja maior ou igual a 17. Se o carteador ultrapassar 21, o jogador ganha. Caso contrário, a mão com pontuação total maior vence. Se o carteador e o jogador tiverem o mesmo total de pontos, o jogo “empatou” e ninguém ganha. A interface gráfica do usuário desse aplicativo pode ser construída com componentes `ImageView`, `TextView` e `Button`.

4.11 (*Aplicativo Blackjack aprimorado*) Aprimore o aplicativo Blackjack do Exercício 4.10, como segue:

- a) Forneça um mecanismo de aposta que permita ao jogador começar com \$1000 e soma ou subtraia desse valor, dependendo de o usuário ganhar ou perder uma mão. Se o jogador ganhar com uma mão que não seja um vinte e um puro, o valor apostado é somado ao total. Se o jogador ganhar com um vinte e um puro, 1,5 vez o valor apostado é somado ao total. Se o jogador perder a mão, o valor apostado é subtraído do total. O jogo termina quando o usuário fica sem dinheiro.
- b) Encontre imagens de fichas de cassino e utilize-as para representar na tela o valor apostado.
- c) Investigue as regras do Blackjack online e forneça recursos para “dobrar”, “desistir” e outros aspectos do jogo.
- d) Alguns cassinos usam variações das regras padrão do Blackjack. Forneça opções que permitam ao usuário escolher as regras sob as quais o jogo deve ser jogado.
- e) Alguns cassinos usam números diferentes de baralhos. Permita que o usuário escolha quantos baralhos devem ser usados.
- f) Permita que o usuário salve o estado do jogo para continuar depois.

4.12 (*Outros aplicativos de jogo de cartas*) Investigue online as regras de qualquer jogo de cartas de sua escolha e implemente o jogo como um aplicativo.

4.13 (*Jogo de cartas Solitaire*) Procure na web as regras de vários jogos de carta do tipo Solitaire (Paciência). Escolha a versão do jogo desejada e implemente-a. (Imagens de cartas são fornecidas com os exemplos do livro.)



Objetivos

Neste capítulo, você vai:

- Usar objetos **Fragment** para fazer melhor uso do espaço útil disponível na tela na interface gráfica do usuário de uma **Activity** em telefones e tablets.
- Exibir um menu de opções na barra de ação para permitir aos usuários configurar as preferências do aplicativo.
- Usar um objeto **PreferenceFragment** para gerenciar automaticamente e persistir as preferências do usuário de um aplicativo.
- Usar as subpastas **assets** de um aplicativo para organizar recursos de imagem e manipulá-los com um componente **AssetManager**.
- Definir uma animação e aplicá-la a um elemento **View**.
- Usar um componente **Handler** a fim de agendar uma tarefa futura para execução na thread da interface gráfica do usuário.
- Usar um componente **Toast** para exibir brevemente mensagens para o usuário.
- Ativar uma atividade específica com um objeto **Intent** explícito.
- Usar várias coleções do pacote **java.util**.
- Definir layouts para várias orientações de dispositivo.
- Usar o mecanismo de log do Android para registrar mensagens de erro.

- 5.1 Introdução
- 5.2 Teste do aplicativo **Flag Quiz**
 - 5.2.1 Importação e execução do aplicativo
 - 5.2.2 Configuração do teste
 - 5.2.3 O teste
- 5.3 Visão geral das tecnologias
 - 5.3.1 Menus
 - 5.3.2 Fragmentos
 - 5.3.3 Métodos do ciclo de vida de um fragmento
 - 5.3.4 Gerenciamento de fragmentos
 - 5.3.5 Preferências
 - 5.3.6 Pasta assets
 - 5.3.7 Pastas de recurso
 - 5.3.8 Suporte para diferentes tamanhos e resoluções de tela
 - 5.3.9 Determinação do tamanho da tela
 - 5.3.10 Componentes Toast para exibir mensagens
 - 5.3.11 Uso de um objeto `Handler` para executar um objeto `Runnable` no futuro
 - 5.3.12 Aplicação de uma animação a um objeto `View`
 - 5.3.13 Registro de mensagens de exceção
 - 5.3.14 Uso de um objeto `Intent` explícito para ativar outra atividade no mesmo aplicativo
 - 5.3.15 Estruturas de dados em Java
- 5.4 Construção da interface gráfica do usuário e do arquivo de recursos
 - 5.4.1 Criação do projeto
 - 5.4.2 `strings.xml` e recursos de `String` formatados
 - 5.4.3 `arrays.xml`
 - 5.4.4 `colors.xml`
 - 5.4.5 `dimens.xml`
 - 5.4.6 Layout de `activity_settings.xml`
 - 5.4.7 Layout de `activity_main.xml` para orientação retrato para telefones e tablets
 - 5.4.8 Layout de `fragment_quiz.xml`
 - 5.4.9 Layout de `activity_main.xml` para orientação paisagem para tablets
 - 5.4.10 Arquivo `preferences.xml` para especificar as configurações do aplicativo
 - 5.4.11 Criação da animação da bandeira
- 5.5 Classe `MainActivity`
 - 5.5.1 Instrução `package`, instruções `import` e campos
 - 5.5.2 Método sobrescrito `onCreate` de `Activity`
 - 5.5.3 Método sobrescrito `onStart` de `Activity`
 - 5.5.4 Método sobrescrito `onOptionsItemSelected` de `Activity`
 - 5.5.5 Método sobrescrito `onOptionsItemSelected` de `Activity`
 - 5.5.6 Classe interna anônima que implementa `OnSharedPreferenceChangeListener`
- 5.6 Classe `QuizFragment`
 - 5.6.1 A instrução `package` e as instruções `import`
 - 5.6.2 Campos
 - 5.6.3 Método sobrescrito `onCreateView` de `Fragment`
 - 5.6.4 Método `updateGuessRows`
 - 5.6.5 Método `updateRegions`
 - 5.6.6 Método `resetQuiz`
 - 5.6.7 Método `loadNextFlag`
 - 5.6.8 Método `getCountryName`
 - 5.6.9 Classe interna anônima que implementa `OnClickListener`
 - 5.6.10 Método `disableButtons`
- 5.7 Classe `SettingsFragment`
- 5.8 Classe `SettingsActivity`
- 5.9 `AndroidManifest.xml`
- 5.10 Para finalizar

Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios

5.1 Introdução

O aplicativo **Flag Quiz** testa sua capacidade de identificar corretamente bandeiras de 10 países (Fig. 5.1). Inicialmente, o aplicativo apresenta a imagem de uma bandeira e três componentes `Button` de palpite representando as respostas possíveis – uma corresponde à bandeira e as outras são respostas incorretas, não duplicadas, selecionadas aleatoriamente. O aplicativo exibe o progresso do usuário no teste, exibindo o número da pergunta (até 10) em um componente `TextView` acima da imagem de bandeira atual.



Figura 5.1 Aplicativo **Flag Quiz** sendo executado em um smartphone na orientação retrato.

Conforme você verá, o aplicativo também permite controlar a dificuldade do teste, especificando se vai exibir três, seis ou nove componentes **Button** de palpite e escolhendo as regiões do mundo que devem ser incluídas no teste. Essas opções são exibidas de formas diferentes, de acordo com o dispositivo que está executando o aplicativo e com sua orientação – o aplicativo aceita orientação retrato em *qualquer* dispositivo, mas orientação paisagem apenas em tablets. Na orientação retrato, o aplicativo exibe um *menu de opções* na barra de ação, contendo um item de menu **Settings**. Quando o usuário seleciona esse item, o aplicativo exibe uma atividade para configurar o número de componen-

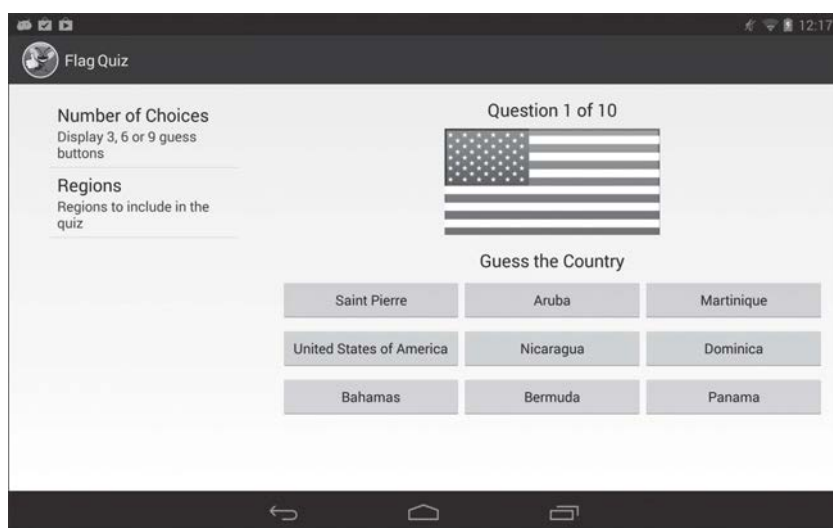


Figura 5.2 Aplicativo **Flag Quiz** sendo executado em um tablet na orientação paisagem.

tes Button de palpite e as regiões do mundo a usar no teste. Em um tablet na orientação paisagem (Fig. 5.2), o aplicativo usa um layout diferente, que exibe suas configurações no lado esquerdo da tela e o teste no lado direito.

Primeiramente, você vai testar o aplicativo. Em seguida, vamos ver um panorama das tecnologias utilizadas para construí-lo. Depois, você vai projetar a interface gráfica do usuário do aplicativo. Por fim, apresentaremos o código-fonte completo do aplicativo e o examinaremos, discutindo os novos recursos do aplicativo com mais detalhes.

5.2 Teste do aplicativo Flag Quiz

Agora você vai testar o aplicativo **Flag Quiz**. Abra o IDE e importe o projeto do aplicativo **Flag Quiz**. Esse aplicativo pode ser testado em um AVD de telefone, em um AVD de tablet ou em um aparelho real. As capturas de tela deste capítulo foram tiradas em um telefone Nexus 4 e em um tablet Nexus 7.

5.2.1 Importação e execução do aplicativo

Execute os passos a seguir para importar o aplicativo para o IDE:

1. **Abra a caixa de diálogo Import.** Selecione **File > Import...**
2. **Importe o projeto do aplicativo Flag Quiz.** Expanda o nó **General** e selecione **Existing Projects into Workspace**. Clique em **Next >** para continuar no passo **Import Projects**. Certifique-se de que **Select root directory** esteja selecionado e, em seguida, clique em **Browse....** Localize a pasta **FlagQuiz** na pasta de exemplos do livro, selecione-a e clique em **OK**. Certifique-se de que **Copy projects into workspace** não esteja selecionado. Clique em **Finish** a fim de importar o projeto para que ele apareça na janela **Package Explorer**.
3. **Ative o aplicativo Flag Quiz.** Clique com o botão direito do mouse no projeto **Flag-Quiz** e selecione **Run As > Android Application** para executar o aplicativo no AVD ou em um dispositivo. Isso constrói o projeto e executa o aplicativo (Fig. 5.1 ou Fig. 5.2).

5.2.2 Configuração do teste

Quando o aplicativo é instalado e executado pela primeira vez, o teste é configurado para exibir três componentes Button de palpite e selecionar bandeiras de *todas* as regiões do mundo. Para este teste, você vai alterar as opções do aplicativo para selecionar bandeiras da América do Norte e vai manter a configuração padrão de três componentes Button de palpite por bandeira. Em um telefone, tablet ou AVD na orientação retrato, toque no ícone do *menu de opções* (Fig. 5.1) na barra de ação para abrir o menu, e selecione **Settings** para ver as opções do aplicativo na tela **Flag Quiz Settings** (Fig. 5.3(a)). Em um tablet ou AVD de tablet na orientação *paisagem*, as opções de configuração do aplicativo aparecem no lado esquerdo da tela (Fig. 5.2). Toque em **Number of Choices** a fim de exibir a caixa de diálogo (Fig. 5.3(b)) para selecionar o número de componentes Button que devem aparecer com cada bandeira. (Em um tablet ou AVD de tablet na orientação paisagem, o aplicativo inteiro fica acinzentado e a caixa de diálogo aparece no centro da tela.) Por padrão, **3** é selecionado. Para tornar o teste mais desafiador, você pode selecionar **6** ou **9** e tocar em **OK**; senão, toque em **Cancel** para voltar à tela **Flag Quiz Settings**. Para este teste, utilizamos a configuração padrão de três componentes Button de palpite.

Em seguida, toque em **Regions** (Fig. 5.4(a)) para exibir as caixas de seleção que representam as regiões do mundo (Fig. 5.4(b)). Por padrão, todas as regiões são habili-

a) Menu com o usuário tocando em **Number of Choices**



Number of Choices selecionado

3 está selecionado, de modo que três componentes Button de palpite serão exibidos com cada bandeira

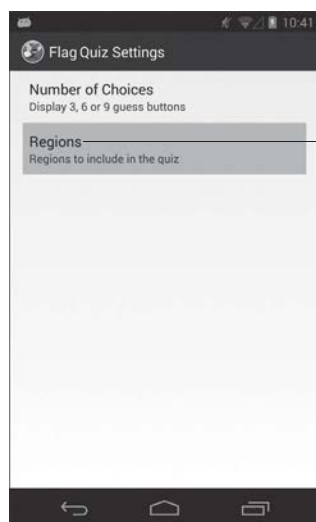
b) Caixa de diálogo mostrando opções para o número de escolhas



Figura 5.3 Tela de configurações do aplicativo **Flag Quiz** e a caixa de diálogo **Number of Choices**.

tadas quando o aplicativo é executado pela primeira vez; portanto, todas as bandeiras do mundo podem ser selecionadas aleatoriamente para o teste. Toque nas caixas de seleção ao lado de **Africa**, **Asia**, **Europe**, **Oceania** e **South America** para desmarcá-las – isso exclui do teste os países dessas regiões. Toque em **OK** para reiniciar o teste com as configurações atualizadas. Em um telefone, tablet ou AVD na orientação retrato, toque no botão *voltar* (←) para voltar ao teste. Em um tablet ou AVD de tablet na orientação paisagem, um teste com as configurações atualizadas aparece imediatamente no lado direito da tela.

a) Menu com o usuário tocando em **Regions**



Regions selecionado

Somente bandeiras de **North America** (América do Norte) serão usadas no teste

b) Caixa de diálogo mostrando as regiões

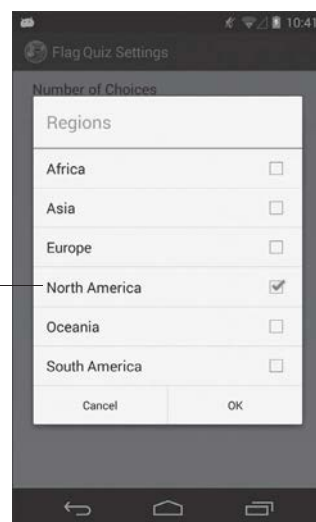


Figura 5.4 Tela de configurações do aplicativo **Flag Quiz** e a caixa de diálogo **Regions**.

5.2.3 O teste

Um novo teste começa com o número de escolhas de resposta selecionadas e bandeiras apenas da região North America. Faça o teste tocando no componente Button de palpite do país que você acha que corresponde a cada bandeira.

Quando se faz uma seleção correta

Se a escolha estiver correta (Fig. 5.5(a)), o aplicativo desabilita todos os componentes Button de resposta e exibe o nome do país em verde, seguido por um ponto de exclamação, na parte inferior da tela (Fig. 5.5(b)). Após uma curta espera, o aplicativo carrega a próxima bandeira e exibe um novo conjunto de componentes Button de resposta.

a) Escolhendo a resposta correta



b) Resposta correta exibida



Todos os componentes **Button** são desabilitados quando o usuário dá um palpite correto

Usuário tocando na resposta correta

A resposta correta aparece em verde

Figura 5.5 O usuário escolhendo a resposta correta e a resposta exibida.

Quando o usuário faz uma seleção incorreta

Se você faz uma seleção incorreta (Fig. 5.6(a)), o aplicativo desabilita o componente Button de nome de país correspondente, usa uma animação para fazer a bandeira *tremular* e exibe **Incorrect!** em vermelho, na parte inferior da tela (Fig. 5.6(b)). Continue arriscando até obter a resposta correta para essa bandeira.

Completando o teste

Depois que você seleciona os 10 nomes de país corretos, um componente AlertDialog pop-up aparece sobre o aplicativo e mostra o número total de palpites e a porcentagem de respostas certas (Fig. 5.7). Quando você toca no componente Button **Reset Quiz** da caixa de diálogo, inicia-se um novo teste com base nas opções atuais.

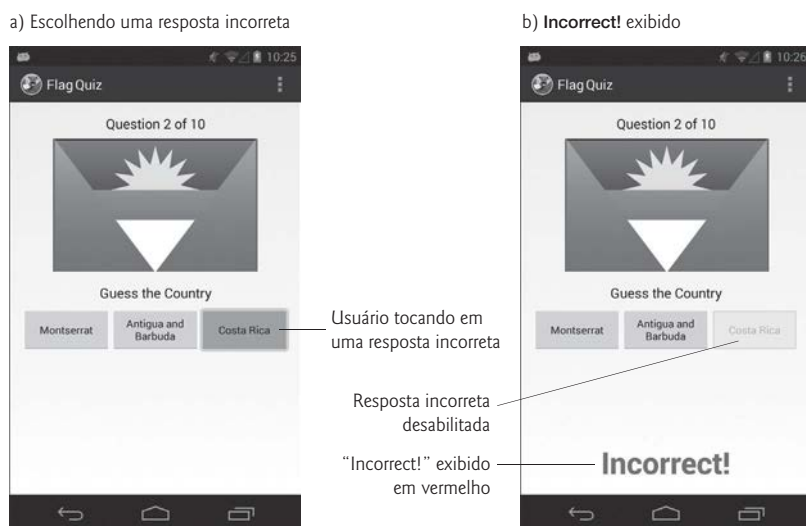


Figura 5.6 Resposta incorreta desabilitada no aplicativo Flag Quiz.

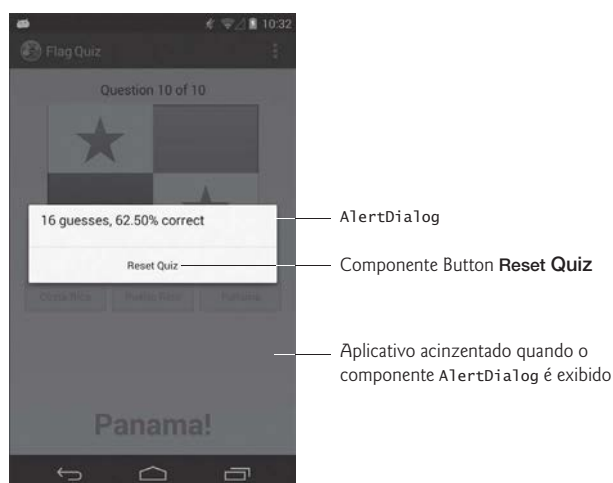


Figura 5.7 Resultados exibidos após a conclusão do teste.

5.3 Visão geral das tecnologias

Esta seção apresenta os recursos que você vai usar para construir o aplicativo Flag Quiz.

5.3.1 Menus

Quando o projeto de um aplicativo é criado no IDE, `MainActivity` é configurado para exibir um menu de opções (☰) no lado direito da barra de ação. O menu contém um item **Settings** que normalmente é usado para mostrar ao usuário as configurações de um aplicativo. Em aplicativos posteriores, você vai aprender a criar mais itens de menu e a decidir quais devem ser exibidos diretamente na barra de ação ou no menu de opções.

O menu de opções é um objeto da classe **Menu** (pacote `android.view`). Para especificar as opções de Menu, sobrescreva o método **onOptionsItemSelected** de Activity (Seção 5.5.4) a fim de adicionar as opções no argumento Menu do método. Quando o usuário seleciona um item de menu, o método **onOptionsItemSelected** de Activity (Seção 5.5.5) responde à seleção.

5.3.2 Fragmentos

Um fragmento normalmente representa uma parte reutilizável da interface do usuário de uma atividade, mas também pode representar uma lógica de programa reutilizável. Este aplicativo utiliza fragmentos para criar e gerenciar partes da interface gráfica do aplicativo. Você pode combinar vários fragmentos para criar interfaces do usuário que tiram proveito dos tamanhos de tela dos tablets. Pode também trocar facilmente os fragmentos para tornar suas interfaces gráficas mais dinâmicas – você vai aprender sobre isso no Capítulo 8.

Fragment (pacote `android.app`) é a classe base de todos os fragmentos. O aplicativo **Flag Quiz** define as seguintes subclasses diretas e indiretas de **Fragment**:

- Classe **QuizFragment** (Seção 5.6) – uma subclasse direta de **Fragment** – exibe a interface gráfica do usuário e define a lógica do teste (*quiz*). Assim como uma atividade, cada objeto **Fragment** tem seu próprio layout, normalmente definido como um recurso de layout, mas pode ser criado dinamicamente. Na Seção 5.4.8, você vai construir a interface gráfica do usuário de **QuizFragment**. Vai usar a classe **QuizFragment** nos layouts de **MainActivity** – uma para dispositivos na orientação retrato e uma para tablets na orientação paisagem.
- A classe **SettingsFragment** (Seção 5.7) é uma subclasse de **PreferenceFragment** (pacote `android.preference`), a qual pode manter automaticamente as preferências do usuário de um aplicativo em um arquivo **SharedPreferences** no dispositivo. Conforme você vai ver, é possível criar um arquivo XML descrevendo as preferências do usuário, e a classe **PreferenceFragment** pode usar esse arquivo XML para construir uma interface gráfica do usuário com as preferências apropriadas (Figs. 5.3 e 5.4).
- Ao terminar um teste, a classe **QuizFragment** cria uma classe interna anônima que estende **DialogFragment** (pacote `android.app`) e a utiliza para exibir um componente **AlertDialog** contendo os resultados do teste (Seção 5.6.9).

Os objetos **Fragment** *devem* ficar em uma atividade – eles não podem ser executados independentemente. Quando este aplicativo é executado na orientação paisagem em um tablet, **MainActivity** armazena todos os objetos **Fragment**. Na orientação retrato (em qualquer dispositivo), **SettingsActivity** (Seção 5.8) armazena o elemento **SettingsFragment**, e **MainActivity** armazena os outros.

Apesar de os objetos **Fragment** terem sido introduzidos no Android 3.0, eles e outros recursos mais recentes do Android podem ser usados em versões anteriores por meio da **Android Support Library**. Para obter mais informações, visite:

<http://developer.android.com/tools/support-library/index.html>

5.3.3 Métodos do ciclo de vida de um fragmento

Como uma atividade, cada fragmento tem um *ciclo de vida* e fornece métodos que podem ser sobrescritos para responder a eventos de ciclo de vida. Neste aplicativo, você vai sobrescrever:

- **onCreate** – Este método (que você vai sobrescrever na classe **SettingsFragment**) é chamado quando um fragmento é criado. **QuizFragment** e **SettingsFragment** são

criados quando os layouts de suas atividades pai são inflados e `DialogFragment`, que exibe os resultados do teste, é criado quando o teste é concluído.

- **onCreateView** – Este método (que você vai sobrescrever na classe `QuizFragment`) é chamado depois de `onCreate` para construir e retornar um objeto `View` contendo a interface gráfica do usuário do fragmento. Conforme você vai ver, este método recebe um objeto **LayoutInflater**, o qual vai ser usado para inflar a interface gráfica do usuário de um fragmento via programação, a partir dos componentes especificados em um layout XML predefinido.

Vamos discutir outros métodos de ciclo de vida de um fragmento à medida que os encontrarmos ao longo do livro. Para ver os detalhes completos do ciclo de vida, visite:

<http://developer.android.com/guide/components/fragments.html>

5.3.4 Gerenciamento de fragmentos

Uma atividade pai gerencia seus fragmentos com um objeto **FragmentManager** (pacote `android.app`), que é retornado pelo método **getFragmentManager** de `Activity`. Se a atividade precisa interagir com um fragmento declarado no layout da atividade e tem uma propriedade `id`, a atividade pode chamar o método **findFragmentById** de `FragmentManager` para obter uma referência para o fragmento especificado. Conforme você vai ver no Capítulo 8, um elemento `FragmentManager` pode usar objetos **FragmentTransactions** (pacote `android.app`) para *adicionar, remover e fazer a transição* entre fragmentos dinamicamente.

5.3.5 Preferências

Na Seção 5.2.2, você alterou as configurações do aplicativo para personalizar o teste. Um elemento `PreferenceFragment` utiliza objetos **Preference** (pacote `android.preference`) para gerenciar essas configurações. Este aplicativo usa a subclasse **ListPreference** de `Preference` para gerenciar o número de componentes `Button` de palpite exibidos para cada bandeira, e a subclasse **MultiSelectListPreference** de `Preference` para gerenciar as regiões do mundo a serem incluídas no teste. Os itens de um elemento `ListPreference` são *mutuamente exclusivos*, ao passo que qualquer número de itens pode ser selecionado em um elemento `MultiSelectListPreference`. Você vai usar um objeto **PreferenceManager** (pacote `android.preference`) para acessar e interagir com as preferências do aplicativo.

5.3.6 Pasta assets

As imagens¹ de bandeira são carregadas no aplicativo somente quando necessário e estão localizadas na **pasta assets** do aplicativo. Para adicionar imagens ao projeto, arrastamos a pasta de cada região a partir de nosso sistema de arquivos para a pasta `assets` no **Package Explorer**. As imagens estão localizadas na pasta `images/FlagQuizImages` com os exemplos do livro.

Ao contrário das pastas `drawable` de um aplicativo, que exigem que seus conteúdos de imagem estejam no nível raiz em cada pasta, a pasta `assets` pode conter arquivos de qualquer tipo e eles podem ser organizados em subpastas – mantemos as imagens das bandeiras de cada região em uma subpasta separada. Os arquivos das subpastas de `assets` são acessados por meio de um componente **AssetManager** (pacote `android.content.res`), o qual pode fornecer uma lista de todos os nomes de arquivo de uma subpasta especificada e pode ser usado para acessar cada `asset`.

¹ Obtivemos as imagens de www.free-country-flags.com.

5.3.7 Pastas de recurso

Na Seção 2.4.4, você aprendeu sobre as subpastas `drawable`, `layout` e `values` da pasta `res` de um aplicativo. Neste aplicativo, vai usar também as pastas de recurso `menu`, `anim` e `xml`. A Figura 5.8 mostra essas subpastas e também as subpastas `animator`, `color` e `raw`.

Subpasta de recurso	Descrição
<code>anim</code>	Nomes de pasta que começam com <code>anim</code> contêm arquivos XML que definem <i>animações com tween</i> , as quais podem mudar a <i>transparência</i> , o <i>tamanho</i> , a <i>posição</i> e a <i>rotação</i> de um objeto ao longo do tempo. Vamos definir uma animação assim na Seção 5.4.11 e, então, executá-la na Seção 5.6.9 para criar um <i>efeito de tremular</i> para dar uma resposta visual ao usuário.
<code>animator</code>	Nomes de pasta que começam com <code>animator</code> contêm arquivos XML que definem <i>animações de propriedade</i> , as quais alteram o valor de uma propriedade de um objeto ao longo do tempo. Em Java, normalmente uma propriedade é implementada em uma classe como uma variável de instância, com métodos de acesso <code>set</code> e <code>get</code> .
<code>color</code>	Nomes de pasta que começam com <code>color</code> contêm arquivos XML que definem uma lista de cores para vários estados, como os estados de um componente <code>Button</code> (<i>não pressionado</i> , <i>pressionado</i> , <i>habilitado</i> , etc.).
<code>raw</code>	Nomes de pasta que começam com <code>raw</code> contêm arquivos de recurso (como clipes de áudio) que são lidos em um aplicativo como fluxos de bytes. Vamos usar esses recursos no Capítulo 6 para reproduzir sons.
<code>menu</code>	Nomes de pasta que começam com <code>menu</code> contêm arquivos XML que descrevem o conteúdo de menus. Quando um projeto é criado, o IDE define automaticamente um menu com uma opção Settings .
<code>xml</code>	Nomes de pasta que começam com <code>xml</code> contêm arquivos XML que não se encaixam nas outras categorias de recurso. Frequentemente, são arquivos de dados XML brutos utilizados pelo aplicativo. Na Seção 5.4.10, você vai criar um arquivo XML que representa as preferências exibidas pelo componente <code>SettingsFragment</code> deste aplicativo.

Figura 5.8 Outras subpastas dentro da pasta `res` de um projeto.

5.3.8 Suporte para diferentes tamanhos e resoluções de tela

Na Seção 2.5.1, você aprendeu que os dispositivos Android têm vários *tamanhos de tela*, *resoluções* e *densidades de pixel* (pontos por polegada ou DPI). Aprendeu também que, normalmente, você fornece imagens e outros recursos visuais em diversas resoluções, para que o Android possa escolher o melhor recurso para a densidade de pixels de um dispositivo. Da mesma forma, na Seção 2.8, você aprendeu a fornecer recursos de string para diferentes idiomas e regiões. O Android utiliza pastas de recurso com *nomes qualificados* para escolher as imagens apropriadas de acordo com a densidade de pixels de um dispositivo, e as strings de idioma corretas de acordo com as configurações de localidade e região do dispositivo. Esse mecanismo também pode ser usado para selecionar recursos de qualquer uma das pastas de recurso discutidas na Seção 5.3.7.

Para a `MainActivity` deste aplicativo, você vai usar qualificadores de tamanho e orientação para determinar o layout a ser usado — um para orientação retrato em telefones e tablets e outro para orientação paisagem apenas em tablets. Para isso, vai definir layouts de `MainActivity`:

- `activity_main.xml`, na pasta `res/layout` do projeto, é o layout padrão.
- `activity_main.xml`, na pasta `res/layout-large-land` do projeto, é usado *apenas* em dispositivos grandes (isto é, tablets), quando o aplicativo está na orientação paisagem (`land`).

Os nomes de pasta de recurso qualificados têm o formato:

nome-qualificadores

onde *qualificadores* consiste em um ou mais qualificadores, separados por traços (-). Existem 18 tipos de qualificadores que podem ser adicionados aos nomes de pasta de recurso. Vamos explicar outros qualificadores à medida que os utilizarmos ao longo do livro. Para ver uma descrição completa de todos os qualificadores de subpasta *res* e as regras para a ordem na qual eles devem ser definidos no nome de uma pasta, visite:

<http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources>

5.3.9 Determinação do tamanho da tela

Neste aplicativo, exibimos o componente *Menu* somente quando ele está sendo executado em um dispositivo do tamanho de um telefone ou quando está sendo executado em um tablet na orientação retrato (Seção 5.5.4). Para determinar isso, vamos usar o componente **WindowManager** (pacote `android.view`) do Android a fim de obter um objeto **Display** que contenha a largura e altura atuais da tela. Isso muda com a orientação do dispositivo – na orientação retrato, a largura do dispositivo é menor que sua altura.

5.3.10 Componentes Toast para exibir mensagens

Um componente **Toast** (pacote `android.widget`) exibe uma mensagem brevemente e, então, desaparece da tela. Eles são frequentemente usados para exibir mensagens de erro secundárias ou mensagens informativas, como uma indicação de que o teste será reiniciado depois que o usuário mudar as preferências do aplicativo. Quando o usuário muda as preferências, exibimos um componente **Toast** para indicar que o teste começará de novo. Também exibimos um componente **Toast** para indicar que pelo menos uma região deve ser selecionada caso o usuário cancele a seleção de todas as regiões – nesse caso, o aplicativo define *North America* como região padrão para o teste.

5.3.11 Uso de um objeto Handler para executar um objeto Runnable no futuro

Quando o usuário dá um palpite correto, o aplicativo exibe a resposta certa por dois segundos, antes de exibir a próxima bandeira. Para isso, usamos um objeto **Handler** (pacote `android.os`). O método **postDelayed** de **Handler** recebe como argumentos um objeto **Runnable** para executar e um valor de tempo de espera em milissegundos. Decorrido o tempo de espera, o objeto **Runnable** de **Handler** é executado na *mesma thread* que criou o objeto **Handler**. *As operações que interagem com a interface gráfica do usuário ou a modificam devem ser efetuadas na thread da interface, pois os componentes da interface não são seguros para threads.*

5.3.12 Aplicação de uma animação a um objeto View

Quando o usuário faz uma escolha incorreta, o aplicativo faz a bandeira tremular ao aplicar um objeto **Animation** (pacote `android.view.animation`) no componente **ImageView**. Usamos o método estático **loadAnimation** de **AnimationUtils** para carregar a animação de um arquivo XML que especifica as opções de animação. Também especificamos o número de vezes que a animação deve se repetir com o método **setRepeatCount** de **Animation** e fazemos a animação chamando o método **startAnimation** de **View** (com o objeto **Animation** como argumento) no componente **ImageView**.

5.3.13 Registro de mensagens de exceção

Quando ocorrem exceções, você pode *registrá-las* para propósitos de depuração com o mecanismo de log interno do Android. O Android fornece a classe **Log** (pacote `android.util`) com vários métodos estáticos que representam mensagens com diversos detalhes. As mensagens registradas podem ser vistas na guia **LogCat**, na parte inferior do IDE, e também com a **ferramenta logcat do Android**. Para saber mais detalhes sobre mensagens de log, visite:

<http://developer.android.com/reference/android/util/Log.html>

5.3.14 Uso de um objeto Intent explícito para ativar outra atividade no mesmo aplicativo

Quando este aplicativo é executado na orientação retrato, as suas preferências aparecem no componente `SettingsActivity` (Seção 5.8). No Capítulo 4, mostramos como usar um objeto *Intent implícito* para exibir uma URL no navegador Web do dispositivo. A Seção 5.5.5 mostra como usar um objeto **Intent explícito** para ativar uma atividade específica no mesmo aplicativo.

5.3.15 Estruturas de dados em Java

Este aplicativo usa várias estruturas de dados do pacote `java.util`. Ele carrega dinamicamente os nomes de arquivo de imagem para as regiões habilitadas e os armazena em um componente `ArrayList<String>`. Usamos o método `shuffle` de `Collections` para tornar a ordem dos nomes de arquivo de imagem aleatória para cada novo jogo. Usamos um segundo componente `ArrayList<String>` para armazenar os nomes de arquivo de imagem dos países para o teste atual. Usamos também um componente **`Set<String>`** para armazenar as regiões do mundo incluídas em um teste. Fazemos referência ao objeto `ArrayList<String>` com uma variável do tipo interface `List<String>` – essa é uma boa prática de programação com Java, que permite alterar estruturas de dados facilmente sem afetar o restante do código de seu aplicativo.

5.4 Construção da interface gráfica do usuário e do arquivo de recursos

Nesta seção, você vai criar o projeto e configurar os recursos de `String`, `array`, `cor`, `dimensão`, `layout` e `animação` utilizados pelo aplicativo **Flag Quiz**.

5.4.1 Criação do projeto

Antes de criar o novo projeto, exclua o projeto **FlagQuiz** que você testou na Seção 5.2 dando um clique nele com o botão direito do mouse e selecionando **Delete**. Na caixa de diálogo que aparece, certifique-se de que **Delete project contents on disk** *não* esteja selecionado e, em seguida, clique em **OK**.

Criação de um novo projeto Blank App

Então, crie um novo **Android Application Project**. Especifique os seguintes valores no primeiro passo de **New Android Application** da caixa de diálogo **New Android Project** e, em seguida, pressione **Next >**:

- **Application name:** Flag Quiz
- **Project name:** FlagQuiz

- **Package name:** com.deitel.flagquiz
- **Minimum Required SDK:** API18: Android 4.3
- **Target SDK:** API19: Android 4.4
- **Compile With:** API19: Android 4.4
- **Theme:** Holo Light with Dark Action Bar

No segundo passo de **New Android Application** da caixa de diálogo **New Android Project**, deixe as configurações padrão e pressione **Next >**. No passo **Configure Launcher Icon**, clique no botão **Browse...**, selecione uma imagem de ícone de aplicativo (fornecida na pasta **images** com os exemplos do livro), pressione **Open** e depois **Next >**. No passo **Create Activity**, selecione **Blank Activity** e pressione **Next >**. No passo **Blank Activity**, deixe as configurações padrão e clique em **Finish** para criar o projeto. Abra **Activity_main.xml** no editor **Graphical Layout** e selecione **Nexus 4** na lista suspensa de tipo de tela. Mais uma vez, usaremos esse dispositivo como base para nosso projeto.

5.4.2 strings.xml e recursos de String formatados

Você criou recursos de String em capítulos anteriores, de modo que mostramos aqui apenas uma tabela (Fig. 5.9) dos nomes dos recursos de String e valores correspondentes. Clique duas vezes em **strings.xml** na pasta **res/values** a fim de exibir o editor de recursos para criar esses recursos de String.

Nome do recurso	Valor
settings_activity	Flag Quiz Settings
number_of_choices	Number of Choices
number_of_choices_description	Display 3, 6 or 9 guess buttons
world_regions	Regions
world_regions_description	Regions to include in the quiz
guess_country	Guess the Country
results	%1\$d guesses, %2\$.02f%% correct
incorrect_answer	Incorrect!
default_region_message	Setting North America as the default region. One region must be selected.
restarting_quiz	Quiz will restart with your new settings
ok	OK
question	Question %1\$d of %2\$d
reset_quiz	Reset Quiz
image_description	Image of the current flag in the quiz
default_region	North_America

Figura 5.9 Recursos de String usados no aplicativo Flag Quiz.

Formate Strings como recursos de String

Os recursos **result** e **question** são componentes Strings de formatação utilizados com o método **format** de String. Quando um recurso String contém vários especificadores de formato, você deve numerá-los para propósitos de adaptação ao idioma. No recurso **results**, a notação **1\$** em **%1\$d** indica que o *primeiro* argumento do método **format** de String após a String de formatação deve substituir o especificador de formato **%1\$d**. Da mesma forma, **2\$** em **%2\$.02f** indica que o *segundo* argumento após a String de formata-

ção deve substituir o especificador de formato `%2$.02f`. O `d` no primeiro especificador de formato indica que estamos formatando um valor inteiro, e o `f` no segundo especificador formata um número de ponto flutuante. Nas versões adaptadas ao idioma de `strings.xml`, os especificadores de formato `%1$d` e `%2$.02f` podem ser reordenados, conforme for necessário, para traduzir corretamente o recurso `String`. O *primeiro* argumento após a `String` de formatação substituirá `%1$d` – independentemente de onde ele apareça na `String` de formatação –, e o *segundo* argumento substituirá `%2$.02f` independentemente de onde apareça na `String` de formatação.

5.4.3 arrays.xml

Na Seção 4.5.8, você criou um recurso de array de `Strings` no arquivo `strings.xml` do aplicativo. Tecnicamente, todos os recursos da pasta `res/values` de seu aplicativo podem ser definidos no *mesmo* arquivo. Contudo, para facilitar o gerenciamento de diferentes tipos de recursos, separamos os arquivos que normalmente são usados para cada um deles. Por exemplo, os recursos de array normalmente são definidos em `arrays.xml`, de cores em `colors.xml`, de `String` em `strings.xml` e de valores numéricos em `values.xml`. Este aplicativo usa três recursos de array de `Strings` que são definidos em `arrays.xml`:

- `regions_list` especifica os nomes das regiões do mundo com as palavras separadas por sublinhados – esses valores são usados para carregar nomes de arquivo de imagem das pastas apropriadas e também como os valores selecionados das regiões do mundo escolhidas pelo usuário no componente `SettingsFragment`.
- `regions_list_for_settings` especifica os nomes das regiões do mundo com as palavras separadas por espaços – esses valores são usados no componente `SettingsFragment` para mostrar os nomes de região para o usuário.
- `guesses_list` especifica as `Strings` 3, 6 e 9 – esses valores são usados no componente `SettingsFragment` para mostrar as opções do número de componentes `Button` de palpite a serem exibidos.

A Figura 5.10 mostra os nomes e valores de elemento desses três recursos de array.

Nome do recurso de array	Valores
<code>regions_list</code>	Africa, Asia, Europe, North_America, Oceania, South_America
<code>regions_list_for_settings</code>	Africa, Asia, Europe, North America, Oceania, South America
<code>guesses_list</code>	3, 6, 9

Figura 5.10 Recursos de array `String` definidos em `arrays.xml`.

Para criar o arquivo e configurar os recursos de array, execute os passos a seguir:

1. Na pasta `res` do projeto, clique com o botão direito do mouse na pasta `values` e, então, selecione **New > Android XML File** para exibir a caixa de diálogo **New Android XML File**. Como você clicou na pasta `values` com o botão direito do mouse, a caixa de diálogo é previamente configurada para adicionar um arquivo de recurso `Values` a essa pasta.
2. Especifique `arrays.xml` no campo **File** e clique em **Finish** para criar o arquivo.
3. Se o IDE abrir o novo arquivo no modo de exibição de XML, clique na guia **Resources**, na parte inferior da janela, para ver o editor de recursos.

4. Para criar um recurso de array de Strings, clique em **Add...**, selecione **String Array** e clique em **OK**.
5. No campo **Name**, digite `regions_list` e salve o arquivo.
6. Selecione o novo recurso de array de Strings e, então, use o botão **Add** para adicionar itens para cada um dos valores mostrados para o array na Fig. 5.10.
7. Repita os passos 4 a 6 para os arrays `regions_list_for_settings` e `guesses_list`. Quando você clicar em **Add...** para criar os recursos de **String Array** adicionais, precisará primeiro selecionar o botão de opção **Create a new element at the top level in Resources**.

5.4.4 colors.xml

Este aplicativo exibe as respostas corretas na cor verde e as incorretas em vermelho. Como qualquer outro recurso, os de cor devem ser definidos em XML para que você possa alterar as cores facilmente, sem modificar o código-fonte Java de seu aplicativo. Normalmente, as cores são definidas em um nome de arquivo `colors.xml`, o qual você precisa criar. Conforme você aprendeu na Seção 3.4.5, as cores são definidas com os esquemas de cor RGB ou ARGB.

Para criar o arquivo e configurar os dois recursos de cor, execute os passos a seguir:

1. Na pasta `res` do projeto, clique com o botão direito do mouse na pasta `values` e, então, selecione **New > Android XML File** para exibir a caixa de diálogo **New Android XML File**.
2. Especifique `colors.xml` no campo **File** e clique em **Finish** para criar o arquivo.
3. Se o IDE abrir o novo arquivo no modo de exibição de XML, clique na guia **Resources**, na parte inferior da janela, para ver o editor de recursos.
4. Para criar um recurso de cor, clique em **Add...**, selecione **Color** e clique em **OK**.
5. Nos campos **Name** e **Value** que aparecem, digite `correct_answer` e `#00CC00`, respectivamente, e salve o arquivo.
6. Repita os passos 4 e 5, mas digite `incorrect_answer` e `#FF0000`.

5.4.5 dimens.xml

Você criou recursos de dimensão em capítulos anteriores, de modo que mostramos aqui apenas uma tabela (Fig. 5.11) dos nomes e valores de recurso de dimensão. Abra `dimens.xml` na pasta `res/values` a fim de exibir o editor de recursos para criar esses recursos. O recurso `spacing` é usado nos layouts como espaçamento entre os vários componentes da interface gráfica do usuário, e o recurso `answer_size` especifica o tamanho da fonte do componente `answerTextView`. Lembre, da Seção 2.5.3, que os tamanhos de fonte devem ser especificados em pixels independentes de escala (sp) para que as fontes em seu aplicativo também possam mudar de escala de acordo com o tamanho de fonte preferido do usuário (conforme especificado nas configurações do dispositivo).

Nome do recurso	Valor
<code>spacing</code>	<code>8dp</code>
<code>answer_size</code>	<code>40sp</code>

Figura 5.11 Recursos de dimensão usados no aplicativo **Flag Quiz**.

5.4.6 Layout de activity_settings.xml

Nesta seção, você vai criar o layout do componente `SettingsActivity` (Seção 5.8) que exibirá o elemento `SettingsFragment` (Seção 5.7). O layout de `SettingsActivity` consistirá apenas em um componente `LinearLayout` contendo a interface gráfica do usuário de `SettingsFragment`. Conforme você vai ver, quando um fragmento é adicionado a um layout, o IDE pode criar a classe do objeto `Fragment` automaticamente. Para criar esse layout, execute os passos a seguir:

1. Na pasta `res` do projeto, clique com o botão direito do mouse em `layout` e selecione **New > Android XML File** para exibir a caixa de diálogo **New Android XML File**. Como você clicou na pasta `layout` com o botão direito do mouse, a caixa de diálogo é previamente configurada para adicionar um arquivo de recurso **Layout**.
2. No campo **File**, digite `activity_settings.xml`.
3. Na seção **Root Element**, selecione **LinearLayout** e clique em **Finish** para criar o arquivo.
4. Da seção **Layouts** da **Palette**, arraste um objeto **Fragment** para a área de projeto ou para o nó `LinearLayout` na janela **Outline**.
5. O passo anterior exibe a caixa de diálogo **Choose Fragment Class**. Se você definiu a classe `Fragment` antes de seu layout, poderia selecionar a classe aqui. Clique em **Create New...** para exibir a caixa de diálogo **New Java Class**.
6. Digite `SettingsFragment` no campo **Name** da caixa de diálogo, mude o valor do campo **Superclass** para `android.preference.PreferenceFragment` e clique em **Finish** para criar a classe. O IDE abre o arquivo Java da classe, o qual você pode fechar por enquanto.
7. Salve `activity_settings.xml`.

5.4.7 Layout de activity_main.xml para orientação retrato para telefones e tablets

Nesta seção, você vai criar o layout do componente `MainActivity` (Seção 5.5) que será utilizado na orientação retrato em todos os dispositivos. O layout da orientação paisagem para tablets vai ser definido na Seção 5.4.9. Esse layout vai exibir apenas o componente `QuizFragment` (Seção 5.6):

1. Na pasta `res/layout` do projeto, abra `activity_main.xml` e siga os passos da Seção 2.5.2 pra trocar de `FrameLayout` para `RelativeLayout`.
2. Da seção **Layouts** da **Palette**, arraste um objeto **Fragment** para o nó `RelativeLayout` na janela **Outline**.
3. Na caixa de diálogo **Choose Fragment Class**, clique em **Create New...** para exibir a caixa de diálogo **New Java Class**.
4. No campo **Name** da caixa de diálogo, digite `QuizFragment` e, então, clique em **Finish** para criar a classe. O IDE abre o arquivo Java da classe, o qual você pode fechar por enquanto.
5. Em `activity_main.xml`, selecione o componente `QuizFragment` na janela **Outline**, configure sua propriedade `id` como `@+id/quizFragment` e, nas propriedades **Layout Parameters**, configure `Width` e `Height` como `match_parent`.
6. Salve `activity_main.xml`.

5.4.8 Layout de fragment_quiz.xml

Normalmente, você define um layout para cada um de seus fragmentos. Para cada layout de fragmento, você vai adicionar um arquivo XML de layout à pasta (ou pastas) `res/layout` de seu aplicativo e especificar a qual classe `Fragment` o layout está associado. Note que não é preciso definir um layout para o componente `SettingsFragment` deste aplicativo, pois sua interface gráfica do usuário é gerada automaticamente pelos recursos herdados da classe `PreferenceFragment`.

Esta seção apresenta o layout de `QuizFragment` (`fragment_quiz.xml`). Você vai definir seu arquivo de layout apenas uma vez na pasta `res/layout` do aplicativo, pois usamos o mesmo layout para `QuizFragment` em todos os dispositivos e orientações de dispositivo. A Figura 5.12 mostra os nomes dos componentes da interface gráfica de usuário do componente `QuizFragment`.



Figura 5.12 Componentes da interface gráfica do aplicativo **Flag Quiz** rotulados com seus valores de propriedade `Id`.

Criando fragment_quiz.xml

Para criar esse `fragment_quiz.xml`, execute os passos a seguir:

1. Na pasta `res` do projeto, clique com o botão direito do mouse na pasta `layout` e, então, selecione **New > Android XML File** para exibir a caixa de diálogo **New Android XML File**.
2. No campo **File**, digite `fragment_quiz.xml`.
3. Na seção **Root Element**, selecione **LinearLayout (Vertical)** e clique em **Finish** para criar o arquivo de layout.
4. Use o editor **Graphical Layout** e a janela **Outline** para formar a estrutura de layout mostrada na Fig. 5.13. À medida que criar os componentes da interface gráfica do usuário, configure suas propriedades `Id`. Para `questionNumberTextView` e `guess-`

CountryTextView, usamos componentes **Medium Text** da seção **Form Widgets** da **Pa-lette**. Para os componentes **Button**, usamos componentes **Small Button**, os quais utilizam um tamanho de fonte menor para que possam encaixar mais texto.

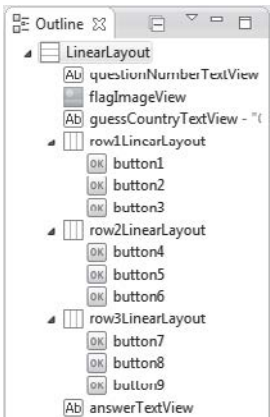


Figura 5.13 Janela **Outline** para `fragment_quiz.xml`.

- 5. Quando terminar o passo 4, configure as propriedades dos componentes da interface gráfica do usuário com os valores mostrados na Fig. 5.14. Configurar a propriedade **Height** de `flagImageView` como `0dp` e a propriedade **Weight** como `1` permite que esse componente seja redimensionado verticalmente para ocupar todo o espaço restante não utilizado pelos outros componentes da interface gráfica. Do mesmo modo, configurar a propriedade **Width** de cada componente **Button** como `0dp` e **Weight** como `1` permite que os componentes **Button** de determinado elemento **LinearLayout** dividam o espaço horizontal igualmente. O valor `fitCenter` de **Scale Type** para `flagImageView` dimensiona a imagem de modo a preencher a largura ou altura do componente `ImageView`, ao passo que mantém a proporção da imagem original. Configurar a propriedade **Adjust View Bounds** de `ImageView` como `true` indica que o componente `ImageView` mantém a proporção de seu elemento `Drawable`.

Componente da interface gráfica do usuário	Propriedade	Valor
questionNumberTextView	Parâmetros do layout	
	Width	wrap_content
	Height	wrap_content
	Gravity	center_horizontal
flagImageView	Outras propriedades	
	Text	@string/question
	Parâmetros do layout	
	Width	wrap_content
	Height	0dp
	Gravity	center

Figura 5.14 Valores de propriedade para os componentes da interface gráfica do usuário em `fragment_quiz.xml`. (continua)

Componente da interface gráfica do usuário	Propriedade	Valor
guessCountryTextView	Weight	1
	Margins	
	Left/Right	@dimen/activity_horizontal_margin
	Top/Bottom	@dimen/activity_vertical_margin
	<i>Outras propriedades</i>	
	Adjust View Bounds	true
	Content Description	@string/image_description
	Scale Type	fitCenter
	<i>Parâmetros do layout</i>	
	Width	wrap_content
LinearLayouts	Height	wrap_content
	Gravity	center_horizontal
	<i>Outras propriedades</i>	
	Text	@string/guess_country
	<i>Parâmetros do layout</i>	
	Width	match_parent
	Height	wrap_content
	Margins	
	Bottom	@dimen/spacing
	<i>Parâmetros do layout</i>	
Buttons	Width	0dp
	Height	fill_parent
	Weight	1
	<i>Parâmetros do layout</i>	
	Width	wrap_content
	Height	wrap_content
	Gravity	center bottom
	<i>Outras propriedades</i>	
	Gravity	center_horizontal
	Text Size	@dimen/answer_size
answerTextView	Text Style	bold

Figura 5.14 Valores de propriedade para os componentes da interface gráfica do usuário em `fragment_quiz.xml`.

5.4.9 Layout de `activity_main.xml` para orientação paisagem para tablets

Na Seção 5.4.7, você definiu o layout da orientação retrato de `MainActivity`, o qual continha apenas o componente `QuizFragment`. Agora, você vai definir o layout da orientação paisagem de `MainActivity` para tablets, o qual conterá os componentes `SettingsFragment` e `QuizFragment`. Para criar o layout, execute os passos a seguir:

1. Clique com o botão direito do mouse na pasta `res` do projeto e, em seguida, selecione **New > Folder**. No campo **Folder name**, digite `layout-large-land` e clique em **Finish**. Os qualificadores `large` e `land` garantem que quaisquer layouts definidos nessa pasta vão ser usados somente em dispositivos grandes, nos quais o aplicativo esteja sendo executado na orientação paisagem.
2. Clique com o botão direito do mouse na pasta `layout-large-land` e selecione **New > Android XML File** para exibir a caixa de diálogo **New Android XML File**; em seguida,

digite `activity_main.xml` no campo **File**. Na seção **Root Element**, selecione **LinearLayout (Horizontal)** e clique em **Finish** para criar o arquivo de layout.

3. Selecione o componente **LinearLayout** e configure sua propriedade **Base Aligned** como **false**.
4. Da seção **Layouts** do editor **Graphical Layout**, arraste um componente **Fragment** para o nó **LinearLayout** na janela **Outline**. Na caixa de diálogo **Choose Fragment Class**, selecione **SettingsFragment** e clique em **OK**.
5. Repita o passo 5, mas selecione **QuizFragment** e clique em **OK**.
6. Selecione o nó **SettingsFragment** na janela **Outline**. Na seção **Layout Parameters**, configure **Width** como **0dp**, **Height** como **match_parent** e **Weight** como **1**.
7. Selecione o nó **QuizFragment** na janela **Outline**. Na seção **Layout Parameters**, configure **Width** como **0dp**, **Height** como **match_parent** e **Weight** como **2**.

Como a propriedade **Weight** de **QuizFragment** é 2 e a de **SettingsFragment** é 1, o componente **QuizFragment** ocupará dois terços do espaço horizontal do layout.

5.4.10 Arquivo `preferences.xml` para especificar as configurações do aplicativo

Nesta seção, você vai criar o arquivo `preferences.xml` utilizado pelo componente **SettingsFragment** para exibir as preferências do aplicativo. Para criar o arquivo:

1. Clique com o botão direito do mouse na pasta `res` e, então, selecione **New > Folder**; no campo **Folder name**, digite `xml` e clique em **Finish**.
2. Clique com o botão direito do mouse na pasta `xml` e selecione **New > Android XML File** para exibir a caixa de diálogo **New Android XML File**.
3. No campo de texto **File**, digite o nome `preferences.xml`.
4. Certifique-se de que **Resource Type** esteja configurado como **Preference** e **Root Element** como **PreferenceScreen**, o que representa uma tela na qual as preferências são exibidas.
5. Clique em **Finish** para criar o arquivo. Se o IDE exibir o código XML bruto, clique na guia **Structure**, na parte inferior da janela, para configurar as preferências.
6. No lado esquerdo da janela, selecione **PreferenceScreen** e clique em **Add...**
7. Na caixa de diálogo que aparece, selecione **ListPreference** e clique em **OK**. Essa preferência exibirá uma lista de opções mutuamente exclusivas.
8. No lado esquerdo da janela, selecione **PreferenceScreen** e clique em **Add...**
9. Na caixa de diálogo que aparece, selecione **MultiSelectListPreference** e clique em **OK**. Essa preferência exibirá uma lista de opções na qual vários itens podem ser selecionados. Todos os itens selecionados são salvos como o valor dessa preferência.
10. Selecione **ListPreference** e, então, configure as propriedades conforme a Fig. 5.15.
11. Selecione **MultiSelectListPreference** e, então, configure as propriedades conforme a Fig. 5.16.
12. Salve e feche `preferences.xml`.

Propriedade	Valor	Descrição
Entries	@array/guesses_list	Array de Strings que será exibida na lista de opções.
Entry values	@array/guesses_list	Array dos valores associados às opções na propriedade Entries . O valor da entrada selecionada será armazenado no elemento <code>SharedPreferences</code> do aplicativo.
Key	pref_numberOfChoices	O nome da preferência armazenada no elemento <code>SharedPreferences</code> do aplicativo.
Title	@string/number_of_choices	O título da preferência exibida na interface gráfica do usuário.
Summary	@string/number_of_choices_description	Descrição resumida da preferência, que é exibida abaixo de seu título.
Persistent	true	Se a preferência deve persistir ou não depois que o aplicativo terminar – se for true, a classe <code>PreferenceFragment</code> faz o valor da preferência persistir imediatamente sempre que ele mudar.
Default value	3	O item da propriedade Entries que é selecionado por padrão.

Figura 5.15 Valores da propriedade `ListPreference`.

Propriedade	Valor	Descrição
Entries	@array/regions_list_for_settings	Array de Strings que será exibida na lista de opções.
Entry values	@array/regions_list	Array dos valores associados às opções na propriedade Entries . Os valores das entradas selecionadas serão <i>todos</i> armazenados no elemento <code>SharedPreferences</code> do aplicativo.
Key	pref_regionsToInclude	O nome da preferência armazenada no elemento <code>SharedPreferences</code> do aplicativo.
Title	@string/world_regions	O título da preferência exibida na interface gráfica do usuário.
Summary	@string/world_regions_description	Descrição resumida da preferência, que é exibida abaixo de seu título.
Persistent	true	Se a preferência deve persistir depois que o aplicativo terminar.
Default value	@array/regions_list	Array dos valores padrão para essa preferência – neste caso, todas as regiões serão selecionadas por padrão.

Figura 5.16 Valores da propriedade `MultiSelectListPreference`.

5.4.11 Criação da animação da bandeira

Nesta seção, você vai criar a animação que faz a bandeira tremular quando o usuário dá um palpite incorreto. Vamos mostrar como essa animação é utilizada pelo aplicativo na Seção 5.6.9. Para criar a animação:

1. Clique com o botão direito do mouse na pasta `res` e, então, selecione **New > Folder**; no campo **Folder name**, digite `anim` e clique em **Finish**.
2. Clique com o botão direito do mouse na pasta `anim` e selecione **New > Android XML File** para exibir a caixa de diálogo **New Android XML File**.
3. No campo de texto **File**, digite o nome `incorrect_shake.xml`.

4. Certifique-se de que **Resource Type** seja **Tween Animation** e de que **Root Element** seja **set**.
5. Clique em **Finish** para criar o arquivo. O arquivo se abre imediatamente no modo de exibição **XML**.
6. Infelizmente, o IDE não fornece um editor para animações; portanto, você precisa modificar o conteúdo XML do arquivo como mostrado na Fig. 5.17.

```

1  <?xml version="1.0" encoding="utf-8"?>
2
3  <set xmlns:android="http://schemas.android.com/apk/res/android"
4      android:interpolator="@android:anim/decelerate_interpolator">
5
6      <translate android:fromXDelta="0" android:toXDelta="-5%p"
7          android:duration="100"/>
8
9      <translate android:fromXDelta="-5%p" android:toXDelta="5%p"
10         android:duration="100" android:startOffset="100"/>
11
12     <translate android:fromXDelta="5%p" android:toXDelta="-5%p"
13         android:duration="100" android:startOffset="200"/>
14 </set>

```

Figura 5.17 Animação de tremular (*incorrect_shake.xml*) aplicada à bandeira quando o usuário dá um palpite incorreto.

Neste exemplo, usamos **animações de View** para criar um *efeito de tremular* que consiste em três animações em um elemento **set de animação** (linhas 3 a 14) – uma coleção de animações que constituem uma animação maior. Os elementos **set de animação** podem conter qualquer combinação de **animações com tween** – **alpha** (transparência), **scale** (redimensionamento), **translate** (movimento) e **rotate** (rotação). Nossa animação de tremular consiste em uma série de três animações **translate**. Uma animação **translate** movimenta um componente **View** dentro de seu componente pai. O Android também suporta *animações de propriedade*, com as quais é possível animar qualquer propriedade de qualquer objeto.

A primeira animação **translate** (linhas 6 e 7) move um componente **View** de um ponto inicial para uma posição final durante um período de tempo especificado. O atributo **android:fromXDelta** é o deslocamento do componente **View** quando a animação começa, e o atributo **android:toXDelta** é o deslocamento desse componente quando a animação termina. Esses atributos podem ter

- valores absolutos (em pixels)
- uma porcentagem do tamanho do componente **View** animado
- uma porcentagem do tamanho do *pai* do componente **View** animado

Para o atributo **android:fromXDelta**, especificamos o valor absoluto 0. Para o atributo **android:toXDelta**, especificamos o valor -5%p, o qual indica que o componente **View** deve se mover para a *esquerda* (devido ao sinal de subtração) por 5% da largura do pai (indicado pelo p). Se quiséssemos mover por 5% da largura do componente **View**, omitiríamos o p. O atributo **android:duration** especifica quanto tempo dura a animação, em milissegundos. Assim, a animação nas linhas 6 e 7 vai mover o componente **View** para a esquerda por 5% da largura de seu pai, em 100 milissegundos.

A segunda animação (linhas 9 e 10) continua a partir de onde a primeira terminou, movendo o componente **View** do deslocamento de -5%p até um deslocamento de 5%p, em 100 milissegundos. Por padrão, as animações de um elemento **set de animação**

são aplicadas simultaneamente (ou seja, em paralelo), mas você pode usar o atributo `android:startOffset` para especificar o número de milissegundos no futuro até que uma animação seja iniciada. Isso pode ser usado para sequenciar as animações em um elemento `set`. Neste caso, a segunda animação começa 100 milissegundos após a primeira. A terceira animação (linhas 12 e 13) é igual à segunda, mas na direção oposta, e começa 200 milissegundos depois da primeira animação.

5.5 Classe MainActivity

A classe `MainActivity` (Figs. 5.18 a 5.23) contém o elemento `QuizFragment` do aplicativo quando este está sendo executado na orientação retrato e contém `SettingsFragment` e (`QuizFragment`) quando está sendo executado em um tablet na orientação paisagem.

5.5.1 Instrução `package`, instruções `import` e campos

A Figura 5.18 mostra a instrução `package`, as instruções `import` e os campos de `MainActivity`. As linhas 6 a 21 importam as diversas classes e interfaces Java e Android utilizadas pelo aplicativo. Realçamos as instruções `import` novas e discutimos as classes e interfaces correspondentes na Seção 5.3 e à medida que forem encontradas nas Seções 5.5.2 a 5.5.6.

```

1 // MainActivity.java
2 // Hospeda o componente QuizFragment em um telefone e os
3 // componentes QuizFragment e SettingsFragment em um tablet
4 package com.deitel.flagquiz;
5
6 import java.util.Set;
7
8 import android.app.Activity;
9 import android.content.Intent;
10 import android.content.SharedPreferences;
11 import android.content.SharedPreferences.OnSharedPreferenceChangeListener;
12 import android.content.pm.ActivityInfo;
13 import android.content.res.Configuration;
14 import android.graphics.Point;
15 import android.os.Bundle;
16 import android.preference.PreferenceManager;
17 import android.view.Display;
18 import android.view.Menu;
19 import android.view.MenuItem;
20 import android.view.WindowManager;
21 import android.widget.Toast;
22
23 public class MainActivity extends Activity
24 {
25     // chaves para ler dados de SharedPreferences
26     public static final String CHOICES = "pref_numberOfChoices";
27     public static final String REGIONS = "pref_regionsToInclude";
28
29     private boolean phoneDevice = true; // usado para impor o modo retrato
30     private boolean preferencesChanged = true; // as preferências mudaram?
31

```

Figura 5.18 Instrução `package`, instruções `import` e campos de `MainActivity`.

As linhas 26 e 27 definem constantes para as chaves de preferência criadas na Seção 5.4.10. Você vai utilizá-las para acessar os valores de preferência correspondentes. A variável booleana `phoneDevice` (linha 29) especifica se o aplicativo está sendo executado em um telefone – em caso positivo, só permitirá a orientação retrato. A variável booleana

preferencesChanged (linha 30) especifica se as preferências do aplicativo mudaram – em caso positivo, o método de ciclo de vida onStart de MainActivity (Seção 5.5.3) chamará os métodos updateGuessRows (Seção 5.6.4) e updateRegions (Seção 5.6.5) de QuizFragment para reconfigurar o teste com base nas novas configurações do usuário. Configuramos essa variável booleana inicialmente como true para que, quando o aplicativo for executado pela primeira vez, o teste seja configurado com as preferências padrão.

5.5.2 Método sobrescrito onCreate de Activity

O método sobrescrito onCreate de Activity (Fig. 5.19) chama setContentView (linha 36) para configurar a interface gráfica do usuário de MainActivity. Se o aplicativo está sendo executado na orientação retrato, o Android escolhe o arquivo activity_main.xml da pasta res/layout; senão, se está sendo executado em um tablet na orientação paisagem, escolhe res/layout-large-land.

```

32  @Override
33  protected void onCreate(Bundle savedInstanceState)
34  {
35      super.onCreate(savedInstanceState);
36      setContentView(R.layout.activity_main);
37
38      // configura valores padrão no elemento SharedPreferences do aplicativo
39      PreferenceManager.setDefaultValues(this, R.xml.preferences, false);
40
41      // registra receptor para alterações em SharedPreferences
42      PreferenceManager.getDefaultSharedPreferences(this).
43          registerOnSharedPreferenceChangeListener(
44              preferenceChangeListener);
45
46      // determina o tamanho da tela
47      int screenSize = getResources().getConfiguration().screenLayout &
48          Configuration.SCREENLAYOUT_SIZE_MASK;
49
50      // se o dispositivo é um tablet, configura phoneDevice como false
51      if (screenSize == Configuration.SCREENLAYOUT_SIZE_LARGE ||
52          screenSize == Configuration.SCREENLAYOUT_SIZE_XLARGE )
53          phoneDevice = false; // não é um dispositivo do tamanho de um telefone
54
55      // se estiver sendo executado em dispositivo do tamanho de um telefone,
56      // só permite orientação retrato
57      if (phoneDevice)
58          setRequestedOrientation(
59              ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
60  } // fim do método onCreate

```

Figura 5.19 Método sobrescrito onCreate de Activity em MainActivity.

Configurando os valores de preferência padrão e registrando um receptor de alteração

Quando o aplicativo é instalado e ativado pela primeira vez, a linha 39 configura as *preferências padrão* chamando o método `setDefaultValues` de `PreferenceManager` – isso cria e inicializa o arquivo `SharedPreferences` do aplicativo usando os valores padrão especificados em `preferences.xml`. O método exige três argumentos:

- o elemento Context das preferências – a atividade (`this`) para a qual você está configurando as preferências padrão;

- o identificador do recurso para o arquivo XML de preferências (`R.xml.preferences`) que você criou na Seção 5.4.10;
- uma variável booleana indicando se os valores padrão devem ser redefinidos sempre que o método `setDefaultValues` for chamado — `false` indica que os valores de preferência padrão devem ser configurados somente na primeira vez que esse método for chamado.

Sempre que o usuário altera as preferências do aplicativo, `MainActivity` deve chamar o método `updateGuessRows` ou `updateRegions` de `QuizFragment` (de acordo com a preferência alterada) para reconfigurar o teste. `MainActivity` registra um elemento `OnSharedPreferenceChangeListener` (linhas 42 a 44) para que seja notificado sempre que uma preferência mudar. O método `getDefaultSharedPreferences` de `PreferenceManager` retorna uma referência para o objeto `SharedPreferences` que representa as preferências do aplicativo, e o método `registerOnSharedPreferenceChangeListener` de `SharedPreferences` registra o receptor (listener), o qual está definido na Seção 5.5.6.

Configurando um telefone para a orientação retrato

As linhas 47 a 53 determinam se o aplicativo está sendo executado em um tablet ou em um telefone. O método herdado `getResources` retorna o objeto `Resources` (pacote `android.content.res`) do aplicativo, que pode ser usado para acessar os seus recursos e descobrir informações sobre o ambiente do aplicativo. O método `getConfiguration` de `Resources` retorna um objeto `Configuration` (pacote `android.content.res`) que contém a variável de instância pública `screenLayout`, a qual você pode usar para determinar a categoria do tamanho da tela do dispositivo. Para isso, primeiro você combina o valor de `screenLayout` com `Configuration.SCREENLAYOUT_SIZE_MASK` usando o operador E bit a bit (`&`). Então, compara o resultado com as constantes de `Configuration` `SCREENLAYOUT_SIZE_LARGE` e `SCREENLAYOUT_SIZE_XLARGE` (linhas 51 e 52). Se uma ou outra corresponder, o aplicativo está sendo executado em um dispositivo com tamanho de tablet. Por fim, se o dispositivo é um telefone, as linhas 57 e 58 chamam o método herdado `setRequestedOrientation` de `Activity` para obrigar o aplicativo a exibir `MainActivity` somente na orientação retrato.

5.5.3 Método sobrescrito `onStart` de `Activity`

O método de ciclo de vida sobrescrito `onStart` de `Activity` (Fig. 5.20) é chamado em dois cenários:

- Quando o aplicativo é executado pela primeira vez, `onStart` é chamado após `onCreate`. Usamos `onStart` neste caso para garantir que o teste seja configurado corretamente, com base nas preferências padrão do aplicativo, quando este for instalado e executado pela primeira vez, ou com base nas preferências atualizadas do usuário, quando for ativado subsequentemente.
- Quando o aplicativo está sendo executado na orientação retrato e o usuário abre a atividade `SettingsActivity`, a `MainActivity` é *pausada*, enquanto `SettingsActivity` é exibido. Quando o usuário retorna para `MainActivity`, `onStart` é chamado novamente. Usamos `onStart` neste caso para garantir que o teste seja reconfigurado corretamente caso o usuário tenha feito quaisquer alterações nas preferências.

Nos dois casos, se `preferencesChanged` for `true`, `onStart` chama os métodos `updateGuessRows` (Seção 5.6.4) e `updateRegions` (Seção 5.6.5) de `QuizFragment` para reconfigurar o teste. Para obter uma referência para `QuizFragment`, a fim de que possamos chamar seus métodos, as linhas 71 e 72 utilizam o método herdado `getFragmentManager` de `Activity`

para obter o elemento `FragmentManager` e, então, chamam seu método `findFragmentById`. Em seguida, as linhas 73 a 76 chamam os métodos `updateGuessRows` e `updateRegions` de `QuizFragment`, passando como argumento o objeto `SharedPreferences` do aplicativo para que esses métodos possam carregar as preferências atuais. A linha 77 reinicia o teste.

```

61 // chamado depois que onCreate completa a execução
62 @Override
63 protected void onStart()
64 {
65     super.onStart();
66
67     if (preferencesChanged)
68     {
69         // agora que as preferências padrão foram configuradas,
70         // inicializa QuizFragment e inicia o teste
71         QuizFragment quizFragment = (QuizFragment)
72             fragmentManager().findFragmentById(R.id.quizFragment);
73         quizFragment.updateGuessRows(
74             PreferenceManager.getDefaultSharedPreferences(this));
75         quizFragment.updateRegions(
76             PreferenceManager.getDefaultSharedPreferences(this));
77         quizFragment.resetQuiz();
78         preferencesChanged = false;
79     }
80 } // fim do método onStart
81

```

Figura 5.20 Método herdado `onStart` de `Activity` em `MainActivity`.

5.5.4 Método sobrescrito `onCreateOptionsMenu` de `Activity`

Sobrescrevemos o método `onCreateOptionsMenu` de `Activity` (Fig. 5.21) para inicializar o menu de opções padrão de `Activity`. O sistema passa o objeto `Menu` onde as opções vão aparecer. Neste caso, queremos mostrar o menu somente quando o aplicativo estiver sendo executado na orientação retrato. As linhas 87 e 88 usam `WindowManager` para obter um objeto `Display` que contém a largura e altura atuais da tela, as quais mudam de acordo com a orientação do dispositivo. Se a largura é menor que a altura, então o dispositivo está na orientação retrato. A linha 89 cria um objeto `Point` para armazenar a largura e altura atuais; então, a linha 90 chama o método `getRealSize` de `Display`, o qual armazena a largura e altura da tela nas variáveis de instância `public x` e `y` (respectivamente) de `Point`. Se a largura é menor que a altura (linha 93), a linha 95 cria o menu a partir de `menu.xml` – o recurso de menu padrão configurado pelo IDE quando você criou o projeto. O método herdado `getMenuInflater` de `Activity` retorna um objeto `MenuInflater`, sobre o qual chamamos `inflate` com dois argumentos – o identificador do recurso de menu que preenche o menu e o objeto `Menu` no qual os itens do menu vão ser colocados. Retornar `true` de `onCreateOptionsMenu` indica que o menu deve ser exibido.

```

82 // mostra o menu se o aplicativo estiver sendo executado em um telefone ou em um
83 // tablet na orientação retrato
84 @Override
85 public boolean onCreateOptionsMenu(Menu menu)
86 {
87     // obtém o objeto Display padrão que representa a tela
88     Display display = ((WindowManager)

```

Figura 5.21 Método sobrescrito `onCreateOptionsMenu` de `Activity` em `MainActivity`.

```

88     getSystemService(WINDOW_SERVICE)).getDefaultDisplay();
89     Point screenSize = new Point(); // usado para armazenar o tamanho da tela
90     display.getRealSize(screenSize); // armazena o tamanho em screenSize
91
92     // só exibe o menu do aplicativo na orientação retrato
93     if (screenSize.x < screenSize.y) // x é a largura, y é a altura
94     {
95         getMenuInflater().inflate(R.menu.main, menu); // infla o menu
96         return true;
97     }
98     else
99         return false;
100 } // fim do método onCreateOptionsMenu
101

```

Figura 5.21 Método sobrescrito `onCreateOptionsMenu` de `Activity` em `MainActivity`.

5.5.5 Método sobrescrito `onOptionsItemSelected` de `Activity`

O método `onOptionsItemSelected` (Fig. 5.22) é chamado quando um item de menu é selecionado. Neste aplicativo, o menu padrão, fornecido pelo IDE quando você criou o projeto, contém apenas o item **Settings**; portanto, se esse método foi chamado, o usuário selecionou **Settings**. A linha 106 cria um objeto `Intent` explícito para ativar a `SettingsActivity`. O construtor de `Intent` usado aqui recebe o objeto `Context`, a partir do qual a atividade vai ser ativada, e a classe que representa essa atividade (`SettingsActivity.class`). Então, passamos esse objeto `Intent` para o método herdado `startActivity` de `Activity` para ativar a atividade.

```

102 // exibe SettingsActivity ao ser executado em um telefone
103 @Override
104 public boolean onOptionsItemSelected(MenuItem item)
105 {
106     Intent preferencesIntent = new Intent(this, SettingsActivity.class);
107     startActivity(preferencesIntent);
108     return super.onOptionsItemSelected(item);
109 }
110

```

Figura 5.22 Método sobrescrito `onOptionsItemSelected` de `Activity` em `MainActivity`.

5.5.6 Classe interna anônima que implementa `OnSharedPreferenceChangeListener`

`preferenceChangeListener` (Fig. 5.23) é um objeto de classe interna anônima que implementa a interface `OnSharedPreferenceChangeListener`. Esse objeto foi registrado no método `onCreate` para detectar as alterações feitas no elemento `SharedPreferences` do aplicativo. Quando uma alteração ocorre, o método `onSharedPreferenceChanged` configura `preferencesChanged` como `true` (linha 120) e depois obtém uma referência para o elemento `QuizFragment` (linhas 122 e 123) para que o teste possa ser reiniciado com as novas preferências. Se a preferência `CHOICES` mudou, as linhas 127 e 128 chamam os métodos `updateGuessRows` e `resetQuiz` de `QuizFragment`.

```

111 // receptor para alterações feitas no elemento SharedPreferences do aplicativo
112 private OnSharedPreferenceChangeListener preferenceChangeListener =
113     new OnSharedPreferenceChangeListener()
114 {
115     // chamado quando o usuário altera as preferências do aplicativo
116     @Override
117     public void onSharedPreferenceChanged(
118         SharedPreferences sharedPreferences, String key)
119     {
120         preferencesChanged = true; // o usuário mudou as configurações do aplicativo
121
122         QuizFragment quizFragment = (QuizFragment)
123             getSupportFragmentManager().findFragmentById(R.id.quizFragment);
124
125         if (key.equals(CHOICES)) // o nº de escolhas a exibir mudou
126         {
127             quizFragment.updateGuessRows(sharedPreferences);
128             quizFragment.resetQuiz();
129         }
130         else if (key.equals(REGIONS)) // as regiões a incluir mudaram
131         {
132             Set<String> regions =
133                 sharedPreferences.getStringSet(REGIONS, null);
134
135             if (regions != null && regions.size() > 0)
136             {
137                 quizFragment.updateRegions(sharedPreferences);
138                 quizFragment.resetQuiz();
139             }
140             else // deve selecionar uma região -- configura North America como padrão
141             {
142                 SharedPreferences.Editor editor = sharedPreferences.edit();
143                 regions.add(
144                     getResources().getString(R.string.default_region));
145                 editor.putStringSet(REGIONS, regions);
146                 editor.commit();
147                 Toast.makeText(MainActivity.this,
148                     R.string.default_region_message,
149                     Toast.LENGTH_SHORT).show();
150             }
151         }
152
153         Toast.makeText(MainActivity.this,
154             R.string.restarting_quiz, Toast.LENGTH_SHORT).show();
155     } // fim do método onSharedPreferenceChanged
156 }; // fim da classe interna anônima
157 } // fim da classe MainActivity

```

Figura 5.23 Classe interna anônima que implementa OnSharedPreferenceChangeListener.

Se a preferência REGIONS mudou, as linhas 132 a 133 obtêm o objeto `Set<String>` contendo as regiões habilitadas. O método `getStringSet` de `SharedPreferences` retorna um objeto `Set<String>` para a chave especificada. O teste deve ter pelo menos uma região habilitada; portanto, se o objeto `Set<String>` não está vazio, as linhas 137 e 138 chamam os métodos `updateRegions` e `resetQuiz` de `QuizFragment`. Caso contrário, as linhas 142 a 146 atualizam a preferência REGIONS, com North America configurado como região padrão, e as linhas 147 a 149 usam um componente `Toast` para indicar que a região padrão foi configurada. O método `makeText` de `Toast` recebe como argumentos o objeto `Context` no qual o componente `Toast` é exibido, a mensagem a ser exibida e a duração da exibição. O método `show` de `Toast` exibe o componen-

te Toast. Independentemente da preferência alterada, as linhas 153 e 154 exibem o componente Toast, indicando que o teste será reiniciado com as novas preferências. A Figura 5.24 mostra o componente Toast que aparece depois que o usuário altera as preferências do aplicativo.



Quiz will restart with your new settings

Figura 5.24 Componente Toast exibido após uma preferência mudar.

5.6 Classe QuizFragment

A classe QuizFragment (Figs. 5.25 a 5.34) constrói a interface gráfica do usuário do aplicativo Flag Quiz e implementa a lógica do teste.

5.6.1 A instrução package e as instruções import

A Figura 5.25 mostra a instrução package e as instruções import de QuizFragment. As linhas 5 a 33 importam as diversas classes e interfaces Java e Android utilizadas pelo aplicativo. Realçamos as instruções import novas e discutimos as classes e interfaces correspondentes na Seção 5.3 e à medida que forem encontradas nas Seções 5.6.2 a 5.6.10.

```

1 // QuizFragment.java
2 // Contém a lógica do aplicativo Flag Quiz
3 package com.deitel.flagquiz;
4
5 import java.io.IOException;
6 import java.io.InputStream;
7 import java.security.SecureRandom;
8 import java.util.ArrayList;
9 import java.util.Collections;
10 import java.util.List;
11 import java.util.Set;
12
13 import android.app.AlertDialog;
14 import android.app.Dialog;
15 import android.app.DialogFragment;
16 import android.app.Fragment;
17 import android.content.DialogInterface;
18 import android.content.SharedPreferences;
19 import android.content.res.AssetManager;
20 import android.graphics.drawable.Drawable;
21 import android.os.Bundle;
22 import android.os.Handler;
23 import android.util.Log;
24 import android.view.LayoutInflater;
25 import android.view.View;
26 import android.view.View.OnClickListener;
27 import android.view.ViewGroup;
28 import android.view.animation.Animation;
29 import android.view.animation.AnimationUtils;
30 import android.widget.Button;
31 import android.widget.ImageView;
32 import android.widget.LinearLayout;
33 import android.widget.TextView;
34

```

Figura 5.25 Instrução package e instruções import de QuizFragment.

5.6.2 Campos

A Figura 5.26 lista as variáveis estáticas e de instância da classe QuizFragment. A constante TAG (linha 38) é usada quando registramos mensagens de erro usando a classe Log (Fig. 5.31) para diferenciar as mensagens de erro dessa atividade das outras que estão sendo gravadas no log do dispositivo. A constante FLAGS_IN_QUIZ (linha 40) representa o número de bandeiras no teste.

```

35 public class QuizFragment extends Fragment
36 {
37     // String usada ao registrar mensagens de erro
38     private static final String TAG = "FlagQuiz Activity";
39
40     private static final int FLAGS_IN_QUIZ = 10;
41
42     private List<String> fileNameList; // nomes de arquivo de bandeira
43     private List<String> quizCountriesList; // países no teste atual
44     private Set<String> regionsSet; // regiões do mundo no teste atual
45     private String correctAnswer; // país correto da bandeira atual
46     private int totalGuesses; // número de palpites dados
47     private int correctAnswers; // número de palpites corretos
48     private int guessRows; // número de linhas exibindo Buttons de palpite
49     private SecureRandom random; // usado para tornar o teste aleatório
50     private Handler handler; // usado para atrasar o carregamento da próxima bandeira
51     private Animation shakeAnimation; // animação para palpite incorreto
52
53     private TextView questionNumberTextView; // mostra o número da pergunta atual
54     private ImageView flagImageView; // exibe uma bandeira
55     private LinearLayout[] guessLinearLayouts; // linhas de Buttons de resposta
56     private TextView answerTextView; // exibe Correct! ou Incorrect!
57

```

Figura 5.26 Campos de QuizFragment.

A variável `fileNameList` (linha 42) armazena os nomes de arquivo de imagem das bandeiras para as regiões geográficas correntemente habilitadas. A variável `quizCountriesList` (linha 43) armazena os nomes de arquivo das bandeiras para os países utilizados no teste atual. A variável `regionsSet` (linha 44) armazena as regiões geográficas que estão habilitadas.

A variável `correctAnswer` (linha 45) armazena o nome de arquivo da bandeira para a resposta correta da bandeira atual. A variável `totalGuesses` (linha 46) armazena o número total de palpites corretos e incorretos dados pelo jogador até o momento. A variável `correctAnswers` (linha 47) é o número de palpites corretos até o momento; isso finalmente será igual a `FLAGS_IN_QUIZ` se o usuário completar o teste. A variável `guessRows` (linha 48) é o número de elementos `LinearLayout` de três componentes `Button` que exibem as escolhas de resposta para a bandeira.

A variável `random` (linha 49) é o gerador de números aleatórios utilizado para selecionar aleatoriamente as bandeiras que serão incluídas no teste e qual dos três componentes `Button` dos elementos `LinearLayout` representa a resposta correta. Quando o usuário seleciona uma resposta correta e o teste não acabou, usamos o elemento `handler` do objeto `Handler` (linha 50) para carregar a próxima bandeira, após um breve espaço de tempo.

O elemento `shakeAnimation` de `Animation` armazena a *animação de tremular*, inflada dinamicamente, que é aplicada à imagem da bandeira quando é dado um palpite incorreto. As linhas 53 a 56 contêm variáveis que usamos para manipular vários componentes da interface gráfica do usuário via programação.

5.6.3 Método sobrescrito onCreateView de Fragment

O método onCreateView de QuizFragment (Fig. 5.27) infla a interface gráfica do usuário e inicializa a maioria das variáveis de instância de QuizFragment – guessRows e regionsSet são inicializadas quando MainActivity chama os métodos updateGuessRows e updateRegions de QuizFragment. Após a chamada do método onCreateView da superclasse (linha 63), inflamos a interface gráfica do usuário de QuizFragment (linhas 64 e 65) usando o objeto LayoutInflater recebido como argumento pelo método onCreateView. O método inflate de LayoutInflater recebe três argumentos:

- o identificador do recurso de layout indicando o layout a ser inflado;
- o elemento ViewGroup (objeto layout) no qual o fragmento será exibido, o qual é recebido como segundo argumento de onCreateView;
- uma variável booleana indicando se a interface gráfica do usuário inflada precisa ou não ser anexada ao componente ViewGroup no segundo argumento – false significa que o fragmento foi declarado no layout da atividade pai, e true indica que você está criando o fragmento dinamicamente e sua interface gráfica deve ser anexada.

O método inflate retorna uma referência para um elemento View que contém a interface gráfica do usuário inflada. Armazenamos isso na variável local view para que possa ser retornada por onCreateView depois que as outras variáveis de instância de QuizFragment forem inicializadas.

```

58 // configura QuizFragment quando sua View é criada
59 @Override
60 public View onCreateView(LayoutInflater inflater, ViewGroup container,
61     Bundle savedInstanceState)
62 {
63     super.onCreateView(inflater, container, savedInstanceState);
64     View view =
65         inflater.inflate(R.layout.fragment_quiz, container, false);
66
67     fileNameList = new ArrayList<String>();
68     quizCountriesList = new ArrayList<String>();
69     random = new SecureRandom();
70     handler = new Handler();
71
72     // carrega a animação de tremular utilizada para respostas incorretas
73     shakeAnimation = AnimationUtils.loadAnimation(getActivity(),
74         R.anim.incorrect_shake);
75     shakeAnimation.setRepeatCount(3); // a animação se repete 3 vezes
76
77     // obtém referências para componentes da interface gráfica do usuário
78     questionNumberTextView =
79         (TextView) view.findViewById(R.id.questionNumberTextView);
80     flagImageView = (ImageView) view.findViewById(R.id.flagImageView);
81     guessLinearLayouts = new LinearLayout[3];
82     guessLinearLayouts[0] =
83         (LinearLayout) view.findViewById(R.id.row1LinearLayout);
84     guessLinearLayouts[1] =
85         (LinearLayout) view.findViewById(R.id.row2LinearLayout);
86     guessLinearLayouts[2] =
87         (LinearLayout) view.findViewById(R.id.row3LinearLayout);
88     answerTextView = (TextView) view.findViewById(R.id.answerTextView);
89

```

Figura 5.27 Método sobrescrito onCreateView de Fragment em QuizFragment. (continua)

```

90         // configura receptores para os componentes Button de palpite
91         for (LinearLayout row : guessLinearLayouts)
92         {
93             for (int column = 0; column < row.getChildCount(); column++)
94             {
95                 Button button = (Button) row.getChildAt(column);
96                 button.setOnClickListener(guessButtonListener);
97             }
98         }
99
100        // configura o texto de questionNumberTextView
101        questionNumberTextView.setText(
102            getResources().getString(R.string.question, 1, FLAGS_IN_QUIZ));
103        return view; // retorna a view do fragmento para exibir
104    } // fim do método onCreateView
105

```

Figura 5.27 Método sobrescrito onCreateView de Fragment em QuizFragment.

As linhas 67 e 68 criam os objetos `ArrayList<String>` que armazenarão os nomes de arquivo de imagem das bandeiras para as regiões geográficas correntemente habilitadas e os nomes dos países do teste atual, respectivamente. A linha 69 cria o objeto `SecureRandom` para tornar aleatórios as bandeiras e os componentes `Button` de palpite do teste. A linha 70 cria o objeto `handler`, o qual vamos usar para atrasar a exibição da próxima bandeira por dois segundos, depois que o usuário der um palpite correto para a bandeira atual.

As linhas 73 e 74 carregam dinamicamente a *animação de tremular* que vai ser aplicada à bandeira quando for dado um palpite incorreto. O método estático `loadAnimation` de `AnimationUtils` carrega a animação do arquivo XML representado pela constante `R.anim.incorrect_shake`. O primeiro argumento indica o objeto `Context` que contém os recursos que serão animados – o método herdado `getActivity` de `Fragment` retorna a classe `Activity` que armazena esse fragmento. `Activity` é uma subclasse indireta de `Context`. A linha 75 especifica o número de vezes que a animação deve se repetir com o método `setRepeatCount` de `Animation`.

As linhas 78 a 88 obtêm referências para vários componentes da interface gráfica de usuário que vamos manipular via programação. As linhas 91 a 98 obtêm cada componente `Button` de palpite dos três elementos `guessLinearLayout` e registram `guessButtonListener` (Seção 5.6.9) como o objeto `OnClickListener`.

As linhas 101 e 102 configuram o texto em `questionNumberTextView` como a `String` retornada pelo método estático `format` de `String`. O primeiro argumento de `format` é o recurso `String R.string.question`, que é a `String` de formatação que contém espaços reservados para dois valores inteiros (conforme descrito na Seção 5.4.2). O método herdado `getResources` de `Fragment` retorna um objeto `Resources` (pacote `android.content.res`) que pode ser usado para carregar recursos. Então, chamamos o método `getString` desse objeto para carregar o recurso `R.string.question`, o qual representa a `String`

```
Question %1$d of %2$d
```

A linha 103 retorna a interface gráfica do usuário de `QuizFragment`.

5.6.4 Método updateGuessRows

O método `updateGuessRows` (Fig. 5.28) é chamado a partir do componente `MainActivity` do aplicativo quando este é ativado e sempre que o usuário muda o número de componentes `Button` de palpite a exibir com cada bandeira. As linhas 110 e 111 utilizam o

argumento `SharedPreferences` do método para obter a `String` da chave `MainActivity.CHOICES` – uma constante contendo o nome da preferência na qual o elemento `SettingsFragment` armazena o número de componentes `Button` de palpite a exibir. A linha 112 converte o valor da preferência para `int` e o divide por 3 para determinar o valor dos elementos `guessRow`, o qual indica quantos dos componentes `guessLinearLayout` devem ser exibidos – cada um com três componentes `Button` de palpite. Em seguida, as linhas 115 e 116 ocultam todos os componentes `guessLinearLayouts`, para que as linhas 119 e 120 possam mostrar os componentes `guessLinearLayouts` apropriados, com base no valor dos elementos `guessRow`.

```

106 // atualiza guessRows com base no valor em SharedPreferences
107 public void updateGuessRows(SharedPreferences sharedPreferences)
108 {
109     // obtém o número de botões de palpite que devem ser exibidos
110     String choices =
111         sharedPreferences.getString(MainActivity.CHOICES, null);
112     guessRows = Integer.parseInt(choices) / 3;
113
114     // oculta todos os componentes LinearLayout de botão de palpite
115     for (LinearLayout layout : guessLinearLayouts)
116         layout.setVisibility(View.INVISIBLE);
117
118     // exibe os componentes LinearLayout de botão de palpite apropriados
119     for (int row = 0; row < guessRows; row++)
120         guessLinearLayouts[row].setVisibility(View.VISIBLE);
121 }
122

```

Figura 5.28 Método `updateGuessRows` de `QuizFragment`.

5.6.5 Método `updateRegions`

O método `updateRegions` (Fig. 5.29) é chamado a partir do componente `MainActivity` do aplicativo quando este é ativado e sempre que o usuário muda as regiões do mundo que devem ser incluídas no teste. As linhas 126 e 127 usam o argumento `SharedPreferences` do método para obter os nomes de todas as regiões habilitadas como um objeto `Set<String>`. `MainActivity.REGIONES` é uma constante contendo o nome da preferência na qual `SettingsFragment` armazena as regiões do mundo habilitadas.

```

123 // atualiza as regiões do mundo para o teste, com base nos valores de
124 // SharedPreferences
125 public void updateRegions(SharedPreferences sharedPreferences)
126 {
127     regionsSet =
128         sharedPreferences.getStringSet(MainActivity.REGIONES, null);
129 }

```

Figura 5.29 Método `updateRegions` de `QuizFragment`.

5.6.6 Método `resetQuiz`

O método `resetQuiz` (Fig. 5.30) prepara e inicia um teste. Lembre-se de que as imagens do jogo estão armazenadas na pasta `assets` do aplicativo. Para acessar o conteúdo dessa pasta, o método obtém o componente `AssetManager` do aplicativo (linha 134) chamando

o método `getAssets` da atividade pai. Em seguida, a linha 135 limpa a lista `fileNameList` a fim de preparar o carregamento dos nomes de arquivo de imagem apenas das regiões geográficas habilitadas. As linhas 140 a 147 iteram por todas as regiões do mundo habilitadas. Para cada uma delas, usamos o método `list` de `AssetManager` (linha 143) para obter um array dos nomes de arquivo de imagem das bandeiras, os quais armazenamos no array `String` `paths`. As linhas 145 e 146 removem a extensão `.png` de cada nome de arquivo e colocam os nomes no componente `fileNameList`. O método `list` de `AssetManager` lança exceções `IOException`, as quais são *verificadas* (portanto, você deve capturar ou declarar a exceção). Se ocorrer uma exceção porque o aplicativo não consegue acessar a pasta `assets`, as linhas 149 a 152 a capturam e a *registram* para propósitos de depuração, com o mecanismo de log interno do Android. O método estático `e` de `Log` é usado para registrar mensagens de erro. Você pode ver uma lista completa de métodos de `Log` em

<http://developer.android.com/reference/android/util/Log.html>

```

130 // prepara e inicia o próximo teste
131 public void resetQuiz()
132 {
133     // usa AssetManager para obter nomes de arquivo de imagem para as regiões
134     // habilitadas
135     AssetManager assets = getActivity().getAssets();
136     fileNameList.clear(); // esvazia a lista de nomes de arquivo de imagem
137
138     try
139     {
140         // faz loop por cada região
141         for (String region : regionsSet)
142         {
143             // obtém uma lista de todos os arquivos de imagem de bandeira nessa região
144             String[] paths = assets.list(region);
145
146             for (String path : paths)
147                 fileNameList.add(path.replace(".png", ""));
148         }
149     } catch (IOException exception)
150     {
151         Log.e(TAG, "Error loading image file names", exception);
152     }
153
154     correctAnswers = 0; // redefine o número de respostas corretas dadas
155     totalGuesses = 0; // redefine o número total de palpites dados pelo usuário
156     quizCountriesList.clear(); // limpa a lista anterior de países do teste
157
158     int flagCounter = 1;
159     int numberOfFlags = fileNameList.size();
160
161     // adiciona FLAGS_IN_QUIZ nomes de arquivo aleatórios a quizCountriesList
162     while (flagCounter <= FLAGS_IN_QUIZ)
163     {
164         int randomIndex = random.nextInt(numberOfFlags);
165
166         // obtém o nome de arquivo aleatório
167         String fileName = fileNameList.get(randomIndex);
168
169         // se a região está habilitada e ainda não foi escolhida
170         if (!quizCountriesList.contains(fileName))

```

Figura 5.30 Método `resetQuiz` de `QuizFragment`.

```

171     {
172         quizCountriesList.add(fileName); // adiciona o arquivo à lista
173         ++flagCounter;
174     }
175 }
176
177 loadNextFlag(); // inicia o teste carregando a primeira bandeira
178 } // fim do método resetQuiz
179

```

Figura 5.30 Método resetQuiz de QuizFragment.

Em seguida, as linhas 154 a 156 redefinem os contadores para o número de palpites corretos dados pelo usuário (correctAnswers) e para o número total de palpites (totalGuesses) como 0 e limpam a lista quizCountriesList.

As linhas 162 a 175 adicionam FLAGS_IN_QUIZ (10) nomes de arquivo selecionados aleatoriamente à lista quizCountriesList. Obtemos o número total de bandeiras e então geramos o índice aleatoriamente, no intervalo de 0 até o número de bandeiras menos um. Usamos esse índice para selecionar um nome de arquivo de imagem da lista fileNameList. Se quizCountriesList ainda não contém esse nome de arquivo, o adicionamos em quizCountriesList e incrementamos flagCounter. Repetimos esse processo até que a quantidade de nomes de arquivo únicos correspondente a FLAGS_IN_QUIZ tenha sido selecionada. Então, a linha 177 chama loadNextFlag (Fig. 5.31) para carregar a primeira bandeira do teste.

5.6.7 Método loadNextFlag

O método loadNextFlag (Fig. 5.31) carrega e exibe a próxima bandeira e o conjunto de componentes Button de resposta correspondente. Os nomes de arquivo de imagem em quizCountriesList têm o formato:

nomeRegião-nomePaís

sem a extensão .png. Se um *nomeRegião* ou um *nomePaís* contém várias palavras, elas são separadas por sublinhados (_).

```

180 // depois que adivinha uma bandeira correta, carrega a próxima bandeira
181 private void loadNextFlag()
182 {
183     // obtém o nome do arquivo da próxima bandeira e o remove da lista
184     String nextImage = quizCountriesList.remove(0);
185     correctAnswer = nextImage; // atualiza a resposta correta
186     answerTextView.setText(""); // limpa answerTextView
187
188     // exibe o número da pergunta atual
189     questionNumberTextView.setText(
190         getResources().getString(R.string.question,
191             (correctAnswers + 1), FLAGS_IN_QUIZ));
192
193     // extrai a região a partir do nome da próxima imagem
194     String region = nextImage.substring(0, nextImage.indexOf('-'));
195
196     // usa AssetManager para carregar a próxima imagem da pasta assets
197     AssetManager assets = getActivity().getAssets();
198

```

Figura 5.31 Método loadNextFlag de QuizFragment. (continua)

```

199     try
200     {
201         // obtém um InputStream para o asset que representa a próxima bandeira
202         InputStream stream =
203             assets.open(region + "/" + nextImage + ".png");
204
205         // carrega o asset como um Drawable e exibe em flagImageView
206         Drawable flag = Drawable.createFromStream(stream, nextImage);
207         flagImageView.setImageDrawable(flag);
208     }
209     catch (IOException exception)
210     {
211         Log.e(TAG, "Error loading " + nextImage, exception);
212     }
213
214     Collections.shuffle(fileNameList); // embaralha os nomes de arquivo
215
216     // coloca a resposta correta no final de fileNameList
217     int correct = fileNameList.indexOf(correctAnswer);
218     fileNameList.add(fileNameList.remove(correct));
219
220     // adiciona 3, 6 ou 9 Buttons de palpite, baseado no valor de guessRows
221     for (int row = 0; row < guessRows; row++)
222     {
223         // coloca componentes Button em currentTableRow
224         for (int column = 0;
225              column < guessLinearLayouts[row].getChildCount(); column++)
226         {
227             // obtém referência para o componente Button a configurar
228             Button newGuessButton =
229                 (Button) guessLinearLayouts[row].getChildAt(column);
230             newGuessButton.setEnabled(true);
231
232             // obtém o nome do país e o configura como o texto de newGuessButton
233             String fileName = fileNameList.get((row * 3) + column);
234             newGuessButton.setText(getCountryName(fileName));
235         }
236     }
237
238     // substitui aleatoriamente um componente Button com a resposta correta
239     int row = random.nextInt(guessRows); // seleciona linha aleatória
240     int column = random.nextInt(3); // seleciona coluna aleatória
241     LinearLayout randomRow = guessLinearLayouts[row]; // obtém a linha
242     String countryName = getCountryName(correctAnswer);
243     ((Button) randomRow.getChildAt(column)).setText(countryName);
244 } // fim do método loadNextFlag
245

```

Figura 5.31 Método loadNextFlag de QuizFragment.

A linha 184 remove o primeiro nome de quizCountriesList e o armazena em nextImage. Também o salvamos em correctAnswer para que possa ser usado posteriormente para determinar se o usuário deu um palpite correto. Em seguida, limpamos o componente answerTextView e exibimos o número da pergunta atual no componente questionNumberTextView (linhas 189 a 191) usando o recurso formatado String R.string.question.

A linha 194 extrai de nextImage a região a ser usada como nome da subpasta assets a partir da qual vamos carregar a imagem. Em seguida, obtemos o componente AssetManager e, então, o utilizamos na instrução try para abrir um elemento InputStream (pacote java.io) para ler os bytes do arquivo da imagem da bandeira. Usamos

esse fluxo (stream) como argumento para o método estático `createFromStream` da classe `Drawable`, o qual cria um objeto `Drawable` (pacote `android.graphics.drawable`). O objeto `Drawable` é configurado como item de `flagImageView` a exibir, chamando seu método `setImageDrawable`. Se ocorrer uma exceção, a registramos para propósitos de depuração (linha 211).

Em seguida, a linha 214 embaralha `fileNameList`, e as linhas 217 e 218 localizam a resposta correta (`correctAnswer`) e movem para o final de `fileNameList` – posteriormente, vamos inserir essa resposta aleatoriamente em um dos componentes `Button` de palpite.

As linhas 221 a 236 iteram pelos componentes `Button` em `guessLinearLayouts` pelo número atual de `guessRows`. Para cada componente `Button`:

- as linhas 228 e 229 obtêm uma referência para o próximo objeto `Button`
- a linha 230 habilita o componente `Button`
- a linha 233 obtém o nome de arquivo da bandeira de `fileNameList`
- a linha 234 configura o texto do componente `Button` com o nome do país retornado pelo método `getCountryName` (Seção 5.6.8)

As linhas 239 a 243 escolhem uma linha (com base no número atual de `guessRows`) e uma coluna aleatórias e, então, configuram o texto do componente `Button` correspondente.

5.6.8 Método `getCountryName`

O método `getCountryName` (Fig. 5.32) obtém o nome do país via parsing, a partir do nome de arquivo de imagem. Primeiramente, obtemos uma substring a partir do traço (-) que separa a região do nome do país. Então, chamamos o método `replace` de `String` para substituir os sublinhados (_) por espaços.

```

246 // obtém o nome do arquivo de bandeira de países via parsing e retorna o nome do país
247 private String getCountryName(String name)
248 {
249     return name.substring(name.indexOf('-') + 1).replace('_', ' ');
250 }
251

```

Figura 5.32 Método `getCountryName` de `QuizFragment`.

5.6.9 Classe interna anônima que implementa `OnClickListener`

As linhas 91 a 98 (Fig. 5.27) registraram `guessButtonListener` (Fig. 5.33) como objeto de tratamento de eventos para cada componente `Button` de palpite. A variável de instância `guessButtonListener` faz referência a um objeto de classe interna anônima que implementa a interface `OnClickListener` para responder a eventos de `Button`. O método recebe o componente `Button` que foi clicado como parâmetro `v`. Obtemos o texto do componente `Button` (linha 259) e o nome do país obtido via parsing (linha 260) e, em seguida, incrementamos `totalGuesses` (o total de palpites).

Se o palpite estiver correto (linha 263), incrementamos `correctAnswers`. Em seguida, configuramos o texto de `answerTextView` com o nome do país e mudamos sua cor para aquela representada pela constante `R.color.correct_answer` (verde) e chamamos nosso método utilitário `disableButtons` (Seção 5.6.10) para desabilitar todos os componentes `Button` de resposta.

```

252 // chamado quando um componente Button de palpite é tocado
253 private OnClickListener guessButtonListener = new OnClickListener()
254 {
255     @Override
256     public void onClick(View v)
257     {
258         Button guessButton = ((Button) v);
259         String guess = guessButton.getText().toString();
260         String answer = getCountryName(correctAnswer);
261         ++totalGuesses; // incrementa o número de palpites dados pelo usuário
262
263         if (guess.equals(answer)) // se o palpite é correto
264         {
265             ++correctAnswers; // incrementa o número de respostas corretas
266
267             // exibe a resposta correta em texto verde
268             answerTextView.setText(answer + "!");
269             answerTextView.setTextColor(
270                 getResources().getColor(R.color.correct_answer));
271
272             disableButtons(); // desabilita todos os componentes Button de palpite
273
274             // se o usuário identificou FLAGS_IN_QUIZ bandeiras corretamente
275             if (correctAnswers == FLAGS_IN_QUIZ)
276             {
277                 // DialogFragment para exibir as estatísticas do teste e iniciar outro
278                 DialogFragment quizResults =
279                     new DialogFragment()
280                     {
281                         // cria um componente AlertDialog e o retorna
282                         @Override
283                         public Dialog onCreateDialog(Bundle bundle)
284                         {
285                             AlertDialog.Builder builder =
286                                 new AlertDialog.Builder(getActivity());
287                             builder.setCancelable(false);
288
289                             builder.setMessage(
290                                 getResources().getString(R.string.results,
291                                     totalGuesses, (1000 / (double) totalGuesses)));
292
293                             // componente Button "Reset Quiz"
294                             builder.setPositiveButton(R.string.reset_quiz,
295                                 new DialogInterface.OnClickListener()
296                                 {
297                                     public void onClick(DialogInterface dialog,
298                                         int id)
299                                     {
300                                         resetQuiz();
301                                     }
302                                 } // fim da classe interna anônima
303                             ); // fim da chamada a setPositiveButton
304
305                             return builder.create(); // retorna o componente AlertDialog
306                         } // fim do método onCreateDialog
307                     }; // fim da classe interna anônima DialogFragment
308
309                 // usa FragmentManager para exibir o componente DialogFragment
310                 quizResults.show(getFragmentManager(), "quiz_results");
311             }
312             else // a resposta está correta, mas o teste não acabou

```

Figura 5.33 Classe interna anônima que implementa OnClickListener. (continua)

```

313         {
314             // carrega a próxima bandeira após um atraso de 2 segundos
315             handler.postDelayed(
316                 new Runnable()
317                 {
318                     @Override
319                     public void run()
320                     {
321                         loadNextFlag();
322                     }
323                 }, 2000); // 2000 milissegundos para um atraso de 2 segundos
324         }
325     }
326     else // o palpite foi incorreto
327     {
328         flagImageView.startAnimation(shakeAnimation); // reproduz o tremular
329
330         // exibe "Incorrect!" em vermelho
331         answerTextView.setText(R.string.incorrect_answer);
332         answerTextView.setTextColor(
333             getResources().getColor(R.color.incorrect_answer));
334         guessButton.setEnabled(false); // desabilita a resposta incorreta
335     }
336 }
337 }; // fim de guessButtonListener
338

```

Figura 5.33 Classe interna anônima que implementa OnClickListener. (continua)

Se `correctAnswers` for igual a `FLAGS_IN_QUIZ` (linha 275), o teste terminou. As linhas 278 a 307 criam uma nova classe interna anônima que estende `DialogFragment` e será usada para exibir os resultados do teste. O método `onCreateDialog` de `DialogFragment` usa um elemento `AlertDialog.Builder` para configurar e criar um componente `AlertDialog` e, então, o retorna. Quando o usuário toca no componente `Button Reset Quiz` da caixa de diálogo, o método `resetQuiz` é chamado para iniciar um novo jogo (linha 300). Para exibir o componente `DialogFragment`, a linha 310 chama seu método `show`, passando como argumentos o elemento `FragmentManager` retornado por `getFragmentManager` e uma `String`. O segundo argumento pode ser usado com o método `getFragmentByTag` de `FragmentManager` para obter uma referência para o componente `DialogFragment` posteriormente – não usamos essa capacidade neste aplicativo.

Se `correctAnswers` for menor do que `FLAGS_IN_QUIZ`, as linhas 315 a 323 chamam o método `postDelayed` do objeto `handler`. O primeiro argumento define uma classe interna anônima que implementa a interface `Runnable` – isso representa a tarefa a ser executada (`loadNextFlag`) após alguns milissegundos no futuro. O segundo argumento é o atraso em milissegundos (2000). Se o palpite estiver incorreto, a linha 328 chama o método `startAnimation` de `flagImageView` para reproduzir a animação `shakeAnimation` que foi carregada no método `onCreateView`. Também configuramos o texto em `answerTextView` para mostrar "Incorrect!" em vermelho (linhas 331 a 333) e, então, desabilitamos o componente `guessButton` correspondente à resposta incorreta.

5.6.10 Método `disableButtons`

O método `disableButtons` (Fig. 5.34) itera pelos componentes `Button` de palpite e os desabilita.

```

339 // método utilitário que desabilita todos os componentes Button de resposta
340 private void disableButtons()
341 {
342     for (int row = 0; row < guessRows; row++)
343     {
344         LinearLayout guessRow = guessLinearLayouts[row];
345         for (int i = 0; i < guessRow.getChildCount(); i++)
346             guessRow.getChildAt(i).setEnabled(false);
347     }
348 }
349 } // fim da classe FlagQuiz

```

Figura 5.34 Método `disableButtons` de `QuizFragment`.

5.7 Classe `SettingsFragment`

A classe `SettingsFragment` (Fig. 5.35) estende `PreferenceFragment`, a qual fornece recursos para gerenciar as configurações do aplicativo. O método sobrescrito `onCreate` (linhas 11 a 16) é chamado quando a classe `SettingsFragment` é criada – ou por `SettingsActivity`, quando o aplicativo está sendo executado na orientação retrato, ou por `MainActivity`, quando está sendo executado em um tablet na orientação paisagem. A linha 15 usa o método herdado `addPreferencesFromResource` de `PreferenceFragment` para construir a interface gráfica de preferências do usuário. O argumento é o identificador de recurso do arquivo `preferences.xml` que você criou na Seção 5.4.10.

```

1 // SettingsFragment.java
2 // Subclasse de PreferenceFragment para gerenciar configurações do aplicativo
3 package com.deitel.flagquiz;
4
5 import android.os.Bundle;
6 import android.preference.PreferenceFragment;
7
8 public class SettingsFragment extends PreferenceFragment
9 {
10     // cria interface gráfica de preferências do usuário a partir do arquivo
11     // preferences.xml em res/xml
12     @Override
13     public void onCreate(Bundle savedInstanceState)
14     {
15         super.onCreate(savedInstanceState);
16         addPreferencesFromResource(R.xml.preferences); // carrega do XML
17     }
18 } // fim da classe SettingsFragment

```

Figura 5.35 Subclasse de `PreferenceFragment` para gerenciar configurações do aplicativo.

5.8 Classe `SettingsActivity`

A classe `SettingsActivity` (Fig. 5.36) armazena o componente `SettingsFragment` quando o aplicativo está sendo executado na orientação retrato. Para criar essa classe, clique com o botão direito do mouse no pacote (`com.deitel.flagquiz`) e selecione **New > Class** para exibir a caixa de diálogo **New Java Class**. Configure o campo **Name** da nova classe como `SettingsActivity`, configure **Superclass** como `android.app.Activity` e clique em **Finish**.

O método sobrescrito `onCreate` (linhas 11 a 16) chama o método `setContentView` de `Activity` para inflar a interface gráfica do usuário definida por `activity_settings.xml` (Seção 5.4.6) – representada pelo recurso `R.layout.activity_settings`.

```

1 // SettingsActivity.java
2 // atividade para exibir SettingsFragment em um telefone
3 package com.deitel.flagquiz;
4
5 import android.app.Activity;
6 import android.os.Bundle;
7
8 public class SettingsActivity extends Activity
9 {
10     // usa fragmentManager para exibir SettingsFragment
11     @Override
12     protected void onCreate(Bundle savedInstanceState)
13     {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_settings);
16     }
17 } // fim da classe SettingsActivity

```

Figura 5.36 Atividade para exibir SettingsFragment em um telefone.

5.9 AndroidManifest.xml

Cada atividade em um aplicativo deve ser declarada no arquivo `AndroidManifest.xml`; caso contrário, o Android não saberá que a atividade existe e não poderá ativá-la. Quando você criou o aplicativo, o IDE declarou seu componente `MainActivity` em `AndroidManifest.xml`. Para declarar o componente `SettingsActivity` do aplicativo:

1. Abra `AndroidManifest.xml` e clique na guia **Application**, na parte inferior do editor de manifesto.
2. Na seção **Application Nodes**, clique em **Add...**, selecione **Activity** na caixa de diálogo que aparece e clique em **OK**.
3. Na seção **Application Nodes**, selecione o novo nó **Activity** para exibir seus atributos na seção **Attributes for Activity**.
4. No campo **Name**, digite `.SettingsActivity`. O ponto (.) antes de `SettingsActivity` é a notação abreviada para o nome de pacote do aplicativo (`com.deitel.flagquiz`).
5. No campo **Label**, digite `@string/settings_activity` – esse recurso de string aparece na barra de ação quando o componente `SettingsActivity` está sendo executado.

Para ver os detalhes completos do arquivo de manifesto, visite <http://developer.android.com/guide/topics/manifest/manifest-intro.html>.

5.10 Para finalizar

Neste capítulo, você construiu o aplicativo **Flag Quiz**, que testa a capacidade de um usuário identificar bandeiras de países corretamente. Um aspecto importante deste aplicativo foi o uso de fragmentos para criar partes da interface gráfica do usuário de uma atividade. Você usou duas atividades para exibir os componentes `QuizFragment` e `SettingsFragment` quando o aplicativo estava sendo executado na orientação retrato, e usou uma atividade para exibir os dois fragmentos quando o aplicativo estava sendo executado em um tablet na orientação paisagem – fazendo, assim, um melhor uso do espaço disponível na tela. Você usou uma subclasse de `PreferenceFragment` para manter automaticamente e fazer persistir

as configurações do aplicativo, e usou uma subclasse de `DialogFragment` para exibir um componente `AlertDialog` para o usuário. Discutimos partes do ciclo de vida de um fragmento e mostramos como usar o componente `FragmentManager` para obter uma referência para um fragmento a fim de que você pudesse interagir com ele via programação.

Na orientação retrato, você usou o menu de ações do aplicativo para permitir ao usuário exibir a atividade `SettingsActivity` contendo o fragmento `SettingsFragment`. Para ativar `SettingsActivity`, você usou um objeto `Intent` explícito.

Mostramos como usar o componente `WindowManager` do Android para obter um objeto `Display` a fim de que você pudesse determinar se o aplicativo estava sendo executado em um tablet na orientação paisagem. Neste caso, você impediu que o menu aparecesse, porque o componente `SettingsFragment` já estava na tela.

Demonstramos como gerenciar um grande número de recursos de imagem utilizando subpastas na pasta `assets` do aplicativo e como acessar esses recursos por meio de um componente `AssetManager`. Você criou um elemento `Drawable` a partir dos bytes de uma imagem, lendo-os de um `InputStream` e, então, exibiu o elemento `Drawable` em um componente `ImageView`.

Você conheceu subpastas adicionais da pasta `res` do aplicativo – menu para armazenar arquivos de recurso de menu, `anim` para armazenar arquivos de recurso de animação e `xml` para armazenar arquivos de dados XML brutos. Discutimos também como usar qualificadores para criar uma pasta para armazenar um layout que só deve ser usado em dispositivos grandes na orientação paisagem.

Você usou objetos `Toast` para exibir mensagens de erro secundárias ou informativas, que aparecem brevemente na tela. Para exibir a próxima bandeira no teste, após uma breve espera, usou um objeto `Handler`, o qual executa um `Runnable` após um número especificado de milissegundos. Você aprendeu que o `Runnable` de um `Handler` é executado na thread que criou o objeto `Handler` (a thread da interface gráfica do usuário, neste aplicativo).

Definimos uma animação em XML e a aplicamos ao componente `ImageView` do aplicativo quando o usuário dava um palpite incorreto, para fornecer a ele uma resposta visual. Você aprendeu a registrar exceções para propósitos de depuração com o mecanismo de `Log` interno do Android. Também usou classes e interfaces adicionais do pacote `java.util`, incluindo `List`, `ArrayList`, `Collections` e `Set`.

No Capítulo 6, você vai criar o aplicativo **Cannon Game** usando múltiplas threads e animação quadro a quadro. Vai manipular gestos de toque para disparar um canhão. Vai também aprender a criar um loop de jogo que atualiza a tela o mais rápido possível para gerar animações suaves e fazer que o jogo pareça ser executado com a mesma rapidez, independentemente da velocidade do processador do dispositivo. Também vamos mostrar como fazer detecção de colisão simples.

Exercícios de revisão

5.1 Preencha os espaços em branco em cada um dos seguintes enunciados:

- a) Para acessar o conteúdo da pasta `assets` do aplicativo, um método deve obter o componente `AssetManager` do aplicativo chamando o método _____ (herdado indiretamente da classe `ContextWrapper`).
- b) Os arquivos das pastas `assets` são acessados por meio de um _____ (pacote `android.content.res`), o qual pode fornecer uma lista de todos os nomes de arquivo de uma subpasta de elementos `assets` especificada e pode ser usado para acessar cada `asset`.

- c) Uma animação _____ movimenta um componente `View` dentro de seu componente pai.
 - d) Por padrão, as animações de um elemento `set` de animação são aplicadas em paralelo, mas você pode usar o atributo _____ para especificar o número de milissegundos no futuro até que uma animação seja iniciada. Isso pode ser usado para sequenciar as animações em um elemento `set`.
- 5.2** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- a) Usamos o método estático `loadAnimation` de `AnimationUtils` para carregar a animação de um arquivo XML que especifica as opções de animação.
 - b) O Android não fornece um mecanismo de registro para propósitos de depuração.
 - c) A propriedade `Adjust View Bounds` de `ImageView` especifica se o componente `ImageView` mantém ou não a proporção de seu elemento `Drawable`.
 - d) Você carrega recursos de array de cor e `String` dos arquivos `colors.xml` e `strings.xml` na memória usando o objeto `Resources` de `Activity`.
 - e) Use atividades para criar componentes reutilizáveis e fazer melhor uso do espaço na tela em um aplicativo para tablets.

Respostas dos exercícios de revisão

- 5.1** a) `getAssets`. b) `AssetManager`. c) `translate`. d) `android:startOffset`.
- 5.2** a) Verdadeira. b) Falsa. Quando ocorrem exceções, você pode registrá-las para propósitos de depuração com os métodos da classe interna `Log`. c) Verdadeira, d) Verdadeira. e) Falsa. Use objetos `Fragment` para criar componentes reutilizáveis e fazer melhor uso do espaço na tela em um aplicativo para tablets.

Exercícios

- 5.3** Preencha os espaços em branco em cada um dos seguintes enunciados:
- a) Quando o usuário seleciona um item em um `Menu`, o método _____ de `Activity` é chamado para responder à seleção.
 - b) Para atrasar uma ação, usamos um objeto _____ (pacote `android.os`) para executar um objeto `Runnable` após uma espera especificada.
 - c) Você pode especificar o número de vezes que a animação deve se repetir com o método _____ de `Animation` e efetuar a animação chamando o método `startAnimation` de `View` (com `Animation` como argumento) no componente `ImageView`.
 - d) Um _____ é uma coleção de animações que constituem uma animação maior.
 - e) O Android suporta animações _____, com as quais é possível animar qualquer propriedade de qualquer objeto.
 - f) Para o atributo `android:fromXDelta`, especificar o valor `-5%p` indica que o componente `view` deve se mover para a _____ por 5% da largura do pai (indicado pelo `p`).
 - g) Usamos o atributo _____ do elemento `application` para aplicar um tema na interface gráfica do aplicativo.
- 5.4** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- a) A classe básica de todos os fragmentos é `BaseFragment` (pacote `android.app`).
 - b) Assim como uma atividade, cada fragmento tem um ciclo de vida.
 - c) Os fragmentos podem ser executados independentemente de uma atividade pai.

Exercícios de projeto

- 5.5 (*Aplicativo Flag Quiz melhorado*)** Faça as seguintes melhorias no aplicativo **Flag Quiz**:
- Conte o número de perguntas respondidas corretamente na primeira tentativa. Depois de respondidas todas as perguntas, exiba uma mensagem descrevendo qual foi o desempenho do usuário nos primeiros palpites.
 - Monitore a contagem à medida que o usuário avança pelo aplicativo. Dê a ele mais pontos por respostas corretas na primeira tentativa, menos por respostas corretas na tentativa seguinte e assim por diante.
 - Use um arquivo `SharedPreferences` para salvar as cinco pontuações mais altas.
 - Acrescente funcionalidade para vários jogadores.
 - Se o usuário adivinhar a bandeira correta, inclua uma “pergunta de bônus”, solicitando o nome da capital do país. Se o usuário responder corretamente na primeira tentativa, some 10 pontos de bônus à pontuação; caso contrário, apenas exiba a resposta correta e permita que o usuário passe para a próxima bandeira.
 - Depois que o usuário responder à pergunta corretamente, inclua um link para esse país na Wikipédia, para que ele possa aprender mais sobre o país à medida que joga. Nesta versão do aplicativo, talvez você queira permitir que o usuário decida quando vai passar para a próxima bandeira.
- 5.6 (*Aplicativo Twitter Searches com fragmentos*)** Reimplemente o aplicativo **Twitter Searches** do Capítulo 4 usando um fragmento. Em vez de fazer a atividade estender `ListActivity`, crie uma subclasse de `ListFragment` e coloque um objeto de sua nova subclasse no elemento `MainActivity` da classe.
- 5.7 (*Aplicativo Road Sign Quiz*)** Crie um aplicativo que teste o conhecimento do usuário sobre sinalização de rodovias. Exiba a imagem de uma sinalização aleatória e peça para o usuário selecionar o nome dela. Visite http://mutcd.fhwa.dot.gov/ser-shs_millennium.htm para ver imagens e informações sobre sinais de trânsito.
- 5.8 (*Aplicativo U.S. State Quiz*)** Usando as técnicas aprendidas neste capítulo, crie um aplicativo que exiba o contorno de um Estado dos Estados Unidos e peça ao usuário para identificá-lo. Se o usuário adivinhar o Estado corretamente, inclua uma “pergunta de bônus”, solicitando o nome da capital dele. Se o usuário responder corretamente, some 10 pontos de bônus à pontuação; caso contrário, apenas exiba a resposta correta e permita que o usuário passe para o próximo Estado. Monitore a pontuação, conforme descrito no Exercício 5.5(c).
- 5.9 (*Aplicativo Country Quiz*)** Usando as técnicas aprendidas neste capítulo, crie um aplicativo que exiba o contorno de um país e peça ao usuário para identificar seu nome. Se o usuário adivinhar o país corretamente, inclua uma “pergunta de bônus”, solicitando o nome da capital. Se o usuário responder corretamente, some 10 pontos de bônus à pontuação; caso contrário, apenas exiba a resposta correta e permita que o usuário passe para o próximo país. Monitore a pontuação, conforme descrito no Exercício 5.5(c).
- 5.10 (*Aplicativo Android Programming Quiz*)** Usando o conhecimento sobre Android que você obteve até aqui, crie um teste de múltipla escolha sobre programação com Android *usando perguntas originais, criadas por você*. Acrescente recursos para vários jogadores para que você possa competir com seus colegas de classe.
- 5.11 (*Aplicativo Movie Trivia Quiz*)** Crie um aplicativo de teste de generalidades sobre cinema.
- 5.12 (*Aplicativo Sports Trivia Quiz*)** Crie um aplicativo de teste de generalidades sobre esportes.
- 5.13 (*Aplicativo Custom Quiz*)** Crie um aplicativo que permita ao usuário criar um teste do tipo verdadeiro/falso ou de múltipla escolha personalizado. Isso é de grande ajuda para os estu-

dos. O usuário pode inserir questões sobre qualquer assunto, incluir as respostas e, então, utilizar o aplicativo para estudar para um teste ou para o exame final.

- 5.14 (*Aplicativo Lottery Number Picker*)** Crie um aplicativo que escolha números de loteria aleatoriamente. Pergunte ao usuário quantos números deve escolher e o número máximo válido na loteria (defina 99 como valor máximo). Forneça cinco combinações de números de loteria possíveis a escolher. Inclua um recurso que permita ao usuário escolher facilmente em uma lista de cinco jogos de loteria populares. Encontre cinco dos jogos de loteria mais populares em sua região e pesquise quantos números devem ser escolhidos para um volante e o número mais alto válido. Permita que o usuário toque no nome do jogo de loteria para escolher números aleatórios para esse jogo.
- 5.15 (*Aplicativo Craps Game*)** Crie um aplicativo que simule um jogo de dados. Nesse jogo, o jogador rola dois dados. Cada dado tem seis faces — imagens de dados são fornecidas com os exemplos do livro. Cada face contém um, dois, três, quatro, cinco ou seis pontos. Depois que os dados param, é calculada a soma dos pontos nas duas faces superiores. Se a soma for 7 ou 11 no primeiro lançamento, o jogador vence. Se a soma for 2, 3 ou 12 no primeiro lançamento (chamado de “craps”), o jogador perde (a “banca” vence). Se a soma for 4, 5, 6, 7, 8, 9 ou 10 no primeiro lançamento, essa soma se torna o “ponto” do jogador. Para vencer, ele deve continuar a lançar os dados até que o valor do ponto seja rolado. O jogador perde rolando um 7 antes de rolar o ponto.
- 5.16 (*Modificação do aplicativo Craps Game*)** Modifique o aplicativo de jogo de dados para permitir apostas. Inicialize a variável `balance` (saldo) com 1000 dólares. Peça ao jogador para digitar uma aposta (*wager*). Verifique se `wager` é menor ou igual a `balance` e, se não for, peça ao usuário para que digite a aposta novamente até que seja inserido um valor válido. Depois de inserida uma aposta correta, execute o jogo de dados apenas uma vez. Se o jogador ganhar, aumente `balance` por `wager` e exiba o novo valor de `balance`. Se o jogador perder, diminua `balance` por `wager`, exiba o novo valor de `balance`, verifique se `balance` se tornou zero e, em caso positivo, exiba a mensagem “Sorry. You busted!” (Desculpe, você faliu!)
- 5.17 (*Aplicativo Computer-Assisted Instruction*)** Crie um aplicativo que ajude um aluno do ensino fundamental a aprender multiplicação. Selecione dois valores inteiros positivos de um dígito. O aplicativo deve então fazer uma pergunta ao usuário, como

Quanto dá 6 vezes 7?

O aluno digita a resposta. Em seguida, o aplicativo verifica a resposta do aluno. Se for correta, exibe uma das seguintes mensagens:

Muito bem!
Excelente!
Bom trabalho!
Continue assim!

e faz outra pergunta. Se a resposta for errada, exibe uma das seguintes mensagens:

Não. Tente outra vez.
Errado. Tente de novo.
Não desista!
Não. Continue tentando.

e permite que o aluno tente a mesma pergunta repetidamente, até acertar. Aprimore o aplicativo para fazer perguntas sobre adição, subtração e divisão.

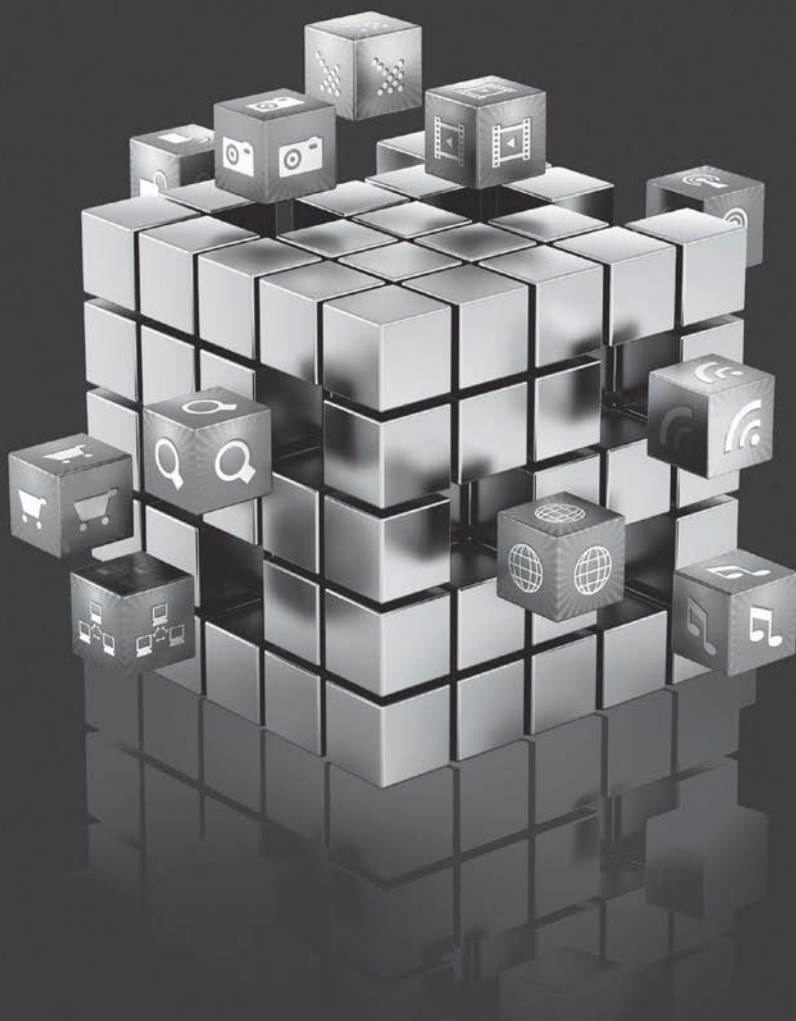
6

Aplicativo Cannon Game

Objetivos

Neste capítulo, você vai:

- Criar um aplicativo de jogo simples, divertido e fácil de codificar.
- Criar uma subclasse personalizada de `SurfaceView` para exibir os elementos gráficos do jogo a partir de uma thread de execução separada.
- Desenhar elementos gráficos usando componentes `Paint` e um `Canvas`.
- Sobrescrever o método `onTouchEvent` de `View` para disparar uma bala de canhão quando o usuário toca na tela.
- Realizar detecção de colisão simples.
- Adicionar som a seu aplicativo usando `SoundPool` e `AudioManager`.
- Sobrescrever os métodos de ciclo de vida `onPause` e `onDestroy` de `Fragment`.



- 6.1 Introdução
- 6.2 Teste do aplicativo **Cannon Game**
- 6.3 Visão geral das tecnologias
 - 6.3.1 Anexação de um componente View personalizado a um layout
 - 6.3.2 Uso da pasta de recurso raw
 - 6.3.3 Métodos de ciclo de vida de Activity e Fragment
 - 6.3.4 Sobrescrevendo o método `onTouchEvent` de View
 - 6.3.5 Adição de som com `SoundPool` e `AudioManager`
 - 6.3.6 Animação quadro a quadro com `Threads`, `SurfaceView` e `SurfaceHolder`
 - 6.3.7 Detecção de colisão simples
 - 6.3.8 Desenho de elementos gráficos com `Paint` e `Canvas`
- 6.4 Construção da interface gráfica do usuário e arquivos de recurso do aplicativo
 - 6.4.1 Criação do projeto
 - 6.4.2 `strings.xml`
 - 6.4.3 `fragment_game.xml`
 - 6.4.4 `activity_main.xml`
 - 6.4.5 Adição dos sons ao aplicativo
- 6.5 A classe `Line` mantém os extremos de uma linha
- 6.6 Subclasse `MainActivity` de `Activity`
- 6.7 Subclasse `CannonGameFragment` de `Fragment`
- 6.8 Subclasse `CannonView` de `View`
 - 6.8.1 As instruções `package` e `import`
 - 6.8.2 Variáveis de instância e constantes
 - 6.8.3 Construtor
 - 6.8.4 Sobrescrevendo o método `onSizeChanged` de `View`
 - 6.8.5 Método `newGame`
 - 6.8.6 Método `updatePositions`
 - 6.8.7 Método `fireCannonball`
 - 6.8.8 Método `alignCannon`
 - 6.8.9 Método `drawGameElements`
 - 6.8.10 Método `showGameOverDialog`
 - 6.8.11 Métodos `stopGame` e `releaseResources`
 - 6.8.12 Implementando os métodos de `SurfaceHolder.Callback`
 - 6.8.13 Sobrescrevendo o método `onTouchEvent` de `View`
 - 6.8.14 `CannonThread`: usando uma `thread` para criar um `loop` de jogo
- 6.9 Para finalizar

Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios

6.1 Introdução

O aplicativo **Cannon Game** o desafia a destruir um alvo de sete partes antes que um limite de 10 segundos expire (Fig. 6.1). O jogo consiste em quatro componentes visuais – um *canhão* controlado por você, uma *bala de canhão*, o *alvo* e uma *barreira* que defende o alvo. Você aponta e dispara o canhão *tocando* na tela – o canhão então mira no ponto tocado e dispara a bala em linha reta nessa direção. Ao final do jogo, o aplicativo exibe um componente `AlertDialog` indicando se você ganhou ou perdeu e mostrando o número de tiros disparados e o tempo decorrido (Fig. 6.2).

O jogo começa com um *limite de tempo de 10 segundos*. Sempre que você destrói uma seção do alvo, três segundos de bônus são *adicionados* ao tempo restante, e sempre que você atinge a barreira, uma penalidade de dois segundos é *subtraída* do tempo restante. Você ganha destruindo todas as seções do alvo antes que o tempo termine – se o cronômetro zerar, você perdeu.

Quando você dispara o canhão, o jogo reproduz um *som de disparo*. Quando a bala do canhão atinge uma parte do alvo, um *som de vidro quebrando* é emitido e essa parte do alvo desaparece. Quando a bala do canhão atinge a barreira, é emitido um *som de golpe* e a bala ricocheteia. A barreira não pode ser destruída. O alvo e a barreira se movem *verticalmente* em velocidades diferentes, mudando de direção quando atingem a parte superior ou inferior da tela.

[Obs.: Devido a problemas de desempenho no Android Emulator, você deve testar este aplicativo em um dispositivo Android.]

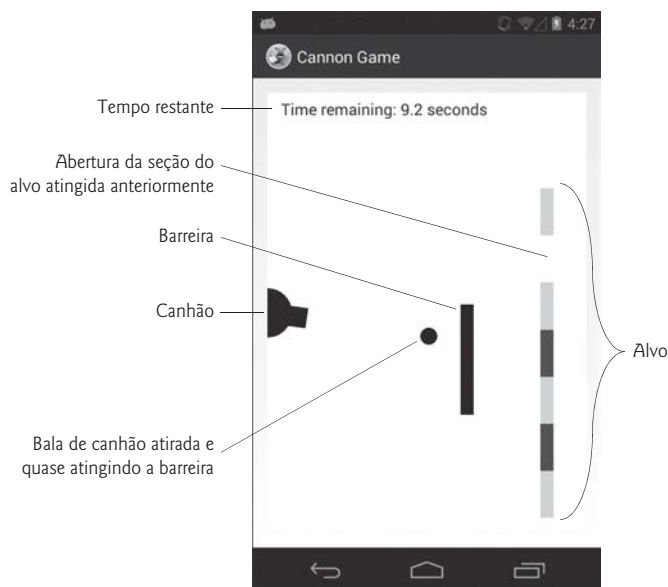
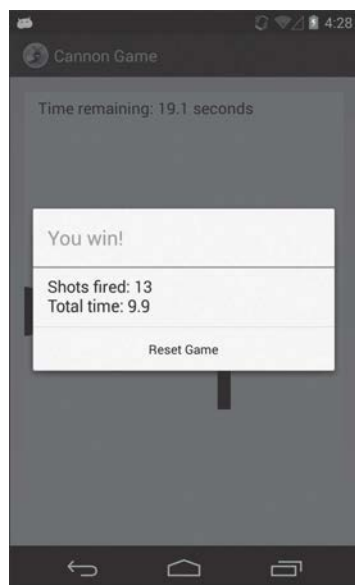


Figura 6.1 Aplicativo **Cannon Game** concluído.

a) Componente `AlertDialog` exibido depois que o usuário destrói todas as sete seções do alvo



b) Componente `AlertDialog` exibido quando o jogo termina antes que o usuário destrua todas as sete seções do alvo

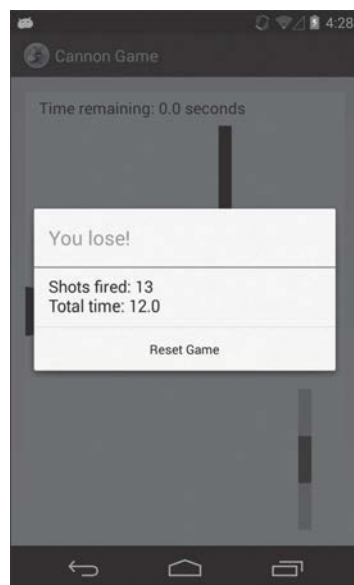


Figura 6.2 Componentes `AlertDialog` do aplicativo **Cannon Game** mostrando uma vitória e uma derrota.

6.2 Teste do aplicativo Cannon Game

Abrindo e executando o aplicativo

Abra o Eclipse e importe o projeto do aplicativo Cannon Game. Execute os passos a seguir:

1. ***Abra a caixa de diálogo Import.*** Selecione File > Import... para abrir a caixa de diálogo Import.
2. ***Importe o projeto do aplicativo Cannon Game.*** Na caixa de diálogo Import, expanda o nó General e selecione Existing Projects into Workspace; em seguida, clique em Next > para passar à etapa Import Projects. Certifique-se de que Select root directory esteja selecionado e, em seguida, clique no botão Browse.... Na caixa de diálogo Browse for Folder, localize a pasta CannonGame na pasta de exemplos do livro, selecione-a e clique em OK. Clique em Finish a fim de importar o projeto para o Eclipse. Agora o projeto aparece na janela Package Explorer, no lado esquerdo da janela do Eclipse.
3. ***Ative o aplicativo Cannon Game.*** No Eclipse, clique com o botão direito do mouse no projeto CannonGame na janela Package Explorer e, em seguida, selecione Run As > Android Application no menu que aparece.

Jogando

Toque na tela para mirar e disparar o canhão. Você só poderá disparar uma bala de canhão se não houver outra na tela. Se estiver executando o jogo em um AVD, seu “dedo” é o mouse. Tente destruir o alvo o mais rápido que puder – o jogo termina se o cronômetro expira ou se você destrói todas as sete partes do alvo.

6.3 Visão geral das tecnologias

Esta seção apresenta as tecnologias novas que usamos no aplicativo Cannon Game, na ordem em que são encontradas no capítulo.

6.3.1 Anexação de um componente View personalizado a um layout

Você pode criar uma visualização *personalizada* estendendo a classe View ou uma de suas subclasses, como fazemos com a classe CannonView (Seção 6.8), a qual estende SurfaceView (discutida em breve). Para adicionar um componente personalizado ao arquivo XML de um layout, você deve *qualificar totalmente* seu nome (isto é, seu nome de pacote e de classe); portanto, a classe do componente View personalizado deve existir antes que se possa adicioná-la ao layout. Demonstramos como criar a classe CannonView e como adicioná-la a um layout na Seção 6.4.3.

6.3.2 Uso da pasta de recurso raw

Os arquivos de mídia, como os sons usados no aplicativo Cannon Game, são colocados na pasta de recursos **res/raw** do aplicativo. A Seção 6.4.5 discute como criar essa pasta. Então, você vai arrastar os arquivos de som do aplicativo para ela.

6.3.3 Métodos de ciclo de vida de Activity e Fragment

Quando um fragmento é anexado a uma atividade, como fizemos no Capítulo 5 e faremos neste, seu ciclo de vida é vinculado ao de sua atividade pai. Existem seis métodos de ciclo de

vida de Activity que têm métodos de ciclo de vida de Fragment correspondentes – onCreate, onStart, onResume, onPause, onStop e onDestroy. Quando o sistema chamar esses métodos em uma atividade, chamará também esses métodos correspondentes (e possivelmente outros métodos de ciclo de vida de Fragment) em todos os fragmentos anexados da atividade.

Este aplicativo utiliza os métodos de ciclo de vida onPause e onDestroy de Fragment. O método **onPause** de uma atividade (Activity) é chamado quando *outra* atividade recebe o foco, o que pausa aquela que perde o foco e a envia para o segundo plano. Quando uma atividade armazena fragmentos e é pausada, os métodos **onPause** de todos os seus fragmentos são chamados. Neste aplicativo, o componente CannonView é exibido em um fragmento chamado CannonGameFragment (Seção 6.7). Sobrescrevemos onPause para suspender a interação no componente CannonView a fim de que o jogo não continue a ser executado quando o usuário não puder interagir com ele – isso economiza a energia da bateria. Muitos métodos de ciclo de vida de Activity têm métodos correspondentes no ciclo de vida de um fragmento.

Quando uma atividade é encerrada, seu método **onDestroy** é chamado, o qual, por sua vez, chama os métodos **onDestroy** de todos os fragmentos armazenados pela atividade. Usamos esse método no componente CannonFragment para *liberar* os recursos de som de CannonView.

Discutiremos outros métodos de ciclo de vida de Activity e Fragment quando forem necessários. Para obter mais informações sobre o ciclo de vida completo de Activity, visite:

<http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle>

e para obter mais informações sobre o ciclo de vida completo de Fragment, visite:

<http://developer.android.com/guide/components/fragments.html#Lifecycle>

6.3.4 Sobrescrevendo o método onTouchEvent de View

Os usuários interagem com este aplicativo tocando na tela do dispositivo. Um *toque* alinha o canhão para o ponto do toque na tela e, então, dispara o canhão. Para processar eventos de toque simples para o componente CannonView, você vai sobrescrever o método **onTouchEvent** de View (Seção 6.8.13) e, então, vai usar constantes da classe **MotionEvent** (pacote android.view) para testar qual tipo de evento ocorreu e processá-lo de forma correspondente.

6.3.5 Adição de som com SoundPool e AudioManager

Os efeitos sonoros de um aplicativo são gerenciados com um objeto **SoundPool** (pacote android.media), o qual pode ser usado para *carregar, reproduzir e descarregar* sons. Os sons são reproduzidos por meio de um dos fluxos (streams) de áudio do Android para *alarmes, música, notificações, toques de telefone, sons do sistema, chamadas telefônicas* e muito mais. A documentação do Android recomenda que os jogos utilizem o *fluxo de áudio de música* para reproduzir sons. Usamos o método **setVolumeControlStream** de Activity para especificar que o volume do jogo pode ser controlado com as teclas de volume do dispositivo. O método recebe uma constante da classe **AudioManager** (pacote android.media), a qual dá acesso aos controles de volume e toque de telefone do dispositivo.

6.3.6 Animação quadro a quadro com Threads, SurfaceView e SurfaceHolder

Este aplicativo *faz suas animações manualmente* ao atualizar os elementos do jogo em uma thread de execução separada. Para isso, usamos uma subclasse de Thread com um método

run que instrui nosso objeto CannonView personalizado a atualizar as posições de todos os elementos do jogo e, então, os desenha. O método run faz as *animações quadro a quadro* – isso é conhecido como **loop do jogo**.

Normalmente, todas as atualizações da interface do usuário de um aplicativo devem ser feitas na thread de execução da interface gráfica do usuário. No Android, é importante minimizar o volume de trabalho feito na thread da interface gráfica para garantir que suas respostas permaneçam rápidas e que não sejam exibidas caixas de diálogo ANR (Application Not Responding). Contudo, os jogos frequentemente exigem lógica complexa que deve ser executada em threads de execução separadas, e essas threads muitas vezes precisam desenhar na tela. Para esses casos, o Android fornece a classe **SurfaceView** – uma subclasse de View na qual uma thread pode desenhar e, então, indicar que os resultados devem ser exibidos na thread da interface gráfica do usuário. Você manipula um elemento SurfaceView por meio de um objeto da classe **SurfaceHolder**, o qual permite obter um elemento Canvas no qual elementos gráficos podem ser desenhados. A classe SurfaceHolder também fornece métodos que dão a uma thread *acesso exclusivo* ao Canvas para desenhar – somente uma thread por vez pode desenhar em um elemento SurfaceView. Cada subclasse de SurfaceView deve implementar a interface **SurfaceHolder.Callback**, a qual contém métodos que são chamados quando o objeto SurfaceView é *criado*, *alterado* (por exemplo, em seu tamanho ou sua orientação) ou *destruído*.

6.3.7 Detecção de colisão simples

O elemento CannonView realiza *detecção de colisão* simples para determinar se a bala do canhão atingiu uma de suas bordas, a barreira ou uma seção do alvo. Essas técnicas são apresentadas na Seção 6.8. Os frameworks de desenvolvimento de jogos normalmente fornecem recursos de detecção de colisão mais sofisticados, “com precisão de pixel”. Estão disponíveis muitos frameworks de desenvolvimento de jogos de código-fonte aberto.

6.3.8 Desenho de elementos gráficos com Paint e Canvas

Usamos métodos da classe **Canvas** (pacote android.graphics) para desenhar texto, linhas e círculos. Os métodos de Canvas desenharam no objeto **Bitmap** de um componente View. Cada método de desenho da classe Canvas utiliza um objeto da classe **Paint** (pacote android.graphics) para especificar as características do desenho, incluindo cor, espessura da linha, tamanho da fonte e muito mais. Esses recursos são apresentados com o método drawGameElements na Seção 6.8. Para obter mais detalhes sobre as características de desenho que podem ser especificadas com um objeto Paint, visite

<http://developer.android.com/reference/android/graphics/Paint.html>

6.4 Construção da interface gráfica do usuário e arquivos de recurso do aplicativo

Nesta seção, você vai criar os arquivos de recurso do aplicativo e o arquivo de layout main.xml.

6.4.1 Criação do projeto

Comece criando um novo projeto Android chamado CannonGame. Especifique os seguintes valores na caixa de diálogo **New Android Project**:

- **Application Name:** Cannon Game
- **Project Name:** CannonGame
- **Package Name:** com.deitel.cannongame
- **Minimum Required SDK:** API18: Android 4.3
- **Target SDK:** API19: Android 4.4
- **Compile With:** API19: Android 4.4
- **Theme:** Holo Light with Dark Action Bar

No segundo passo de **New Android Application** da caixa de diálogo **New Android Project**, deixe as configurações padrão e pressione **Next >**. No passo **Configure Launcher Icon**, selecione uma imagem de ícone de aplicativo e, então, pressione **Next >**. No passo **Create Activity**, selecione **Blank Activity** e pressione **Next >**. No passo **Blank Activity**, deixe as configurações padrão e clique em **Finish** para criar o projeto. Abra `activity_main.xml` no editor **Graphical Layout** e selecione **Nexus 4** na lista suspensa de tipo de tela. Mais uma vez, usaremos esse dispositivo como base para nosso projeto.

Configure o aplicativo para a orientação retrato

O jogo do canhão é projetado para funcionar melhor na orientação retrato. Siga os passos executados na Seção 3.6 para configurar a orientação da tela do aplicativo como retrato.

6.4.2 strings.xml

Você criou recursos de `String` em capítulos anteriores, de modo que mostramos aqui apenas uma tabela (Fig. 6.3) dos nomes dos recursos de `String` e valores correspondentes. Clique duas vezes em `strings.xml` na pasta `res/values` a fim de exibir o editor de recursos para criar esses recursos de `String`.

Nome do recurso	Valor
<code>results_format</code>	<code>Shots fired: %1\$d\nTotal time: %2\$.1f</code>
<code>reset_game</code>	<code>Reset Game</code>
<code>win</code>	<code>You win!</code>
<code>lose</code>	<code>You lose!</code>
<code>time_remaining_format</code>	<code>Time remaining: %.1f seconds</code>

Figura 6.3 Recursos de `String` usados no aplicativo **Cannon Game**.

6.4.3 fragment_game.xml

O layout de `fragment_game.xml` para o componente `CannonGameFragment` contém um objeto `FrameLayout` que exibe o elemento `CannonView`. Um objeto `FrameLayout` é projetado para exibir apenas um componente `View` – neste caso, o elemento `CannonView`. Nesta seção, você vai criar o layout de `CannonGameFragment` e a classe de `CannonView`. Para adicionar o layout de `fragment_game.xml`, execute os passos a seguir:

1. Expanda o nó `res/layout` do projeto no **Package Explorer**.
2. Clique com o botão direito do mouse na pasta `layout` e selecione **New > Android XML File** para exibir a caixa de diálogo **New Android XML File**.
3. No campo **File** da caixa de diálogo, digite `fragment_game.xml`.
4. Na seção **Root Element**, selecione `FrameLayout` e, então, clique em **Finish**.

5. Da seção **Advanced** da **Palette**, arraste um objeto **view** (com **v** minúsculo) para a área de projeto.
6. O passo anterior exibe a caixa de diálogo **Choose Custom View Class**. Nessa caixa de diálogo, clique em **Create New...** para exibir a caixa de diálogo **New Java Class**.
7. No campo **Name**, digite **CannonView**. No campo **Superclass**, mude a superclasse de **android.view.View** para **android.view.SurfaceView**. Certifique-se de que **Constructors from superclass** esteja selecionado e, em seguida, clique em **Finish**. Isso cria e abre **CannonView.java**. Vamos usar somente o construtor com dois argumentos; portanto, exclua os outros dois. Salve e feche **CannonView.java**.
8. Em **fragment_game.xml**, selecione **view1** na janela **Outline**. Na seção **Layout Parameters** da janela **Properties**, configure **Width** e **Height** como **match_parent**.
9. Na janela **Outline**, clique com o botão direito do mouse em **view1**, selecione **Edit ID...**, mude o nome de **view1** para **cannonView** e clique em **OK**.
10. Salve **fragment_game.xml**.

6.4.4 activity_main.xml

O layout de **activity_main.xml** para o componente **MainActivity** deste aplicativo contém apenas o elemento **CannonGameFragment**. Para adicionar esse fragmento ao layout:

1. Abra **activity_main.xml** no editor **Graphical Layout** e siga os passos da Seção 2.5.2 para mudar de **FrameLayout** para **RelativeLayout**.
2. Da seção **Layouts** da **Palette**, arraste um objeto **Fragment** para a área de projeto ou para o nó **RelativeLayout** na janela **Outline**.
3. O passo anterior exibe a caixa de diálogo **Choose Fragment Class**. Clique em **Create New...** para exibir a caixa de diálogo **New Java Class**.
4. Digite **CannonGameFragment** no campo **Name** da caixa de diálogo, mude o valor do campo **Superclass** para **android.app.Fragment** e clique em **Finish** para criar a classe. O IDE abre o arquivo Java da classe, o qual você pode fechar por enquanto.
5. Salve **activity_main.xml**.

6.4.5 Adição dos sons ao aplicativo

Conforme mencionamos anteriormente, os arquivos de som estão armazenados na pasta **res/raw** do aplicativo. Este aplicativo usa três arquivos de som – **blocker_hit.wav**, **target_hit.wav** e **cannon_fire.wav** –, os quais se encontram com os exemplos do livro na pasta **sounds**. Para adicionar esses arquivos em seu projeto:

1. Clique com o botão direito do mouse na pasta **res** do aplicativo e, em seguida, selecione **New > Folder**.
2. Especifique o nome de pasta **raw** e clique em **Finish** para criar a pasta.
3. Arraste os arquivos de som para a pasta **res/raw**.

6.5 A classe **Line** mantém os extremos de uma linha

Este aplicativo consiste em quatro classes:

- **Line** (Fig. 6.4)

- MainActivity (a subclasse de Activity; Seção 6.6)
- CannonGameFragment (Seção 6.7)
- CannonView (Seção 6.8)

Nesta seção, discutimos a classe `Line`, a qual representa os componentes `Point` inicial e final de uma linha. Os objetos dessa classe definem a barreira e o alvo do jogo. Para adicionar a classe `Line` no projeto:

1. Expanda o nó `src` do projeto no **Package Explorer**.
2. Clique com o botão direito do mouse no pacote (`com.deitel.cannongame`) e selecione **New > Class** para exibir a caixa de diálogo **New Java Class**.
3. No campo **Name** da caixa de diálogo, digite `Line` e clique em **Finish**.
4. Insira o código da Figura 6.4 no arquivo `Line.java`. O construtor padrão de `Point` configura as variáveis de instância `public x` e `y` de `Point` como 0.

```
1 // Line.java
2 // A classe Line representa uma linha com dois pontos extremos.
3 package com.deitel.cannongame;
4
5 import android.graphics.Point;
6
7 public class Line
8 {
9     public Point start = new Point(); // objeto Point inicial -- (0,0) por padrão
10    public Point end = new Point(); // objeto Point final -- (0,0) por padrão
11 } // fim da classe Line
```

Figura 6.4 A classe `Line` representa uma linha com dois pontos extremos.

6.6 Subclasse `MainActivity` de `Activity`

A classe `MainActivity` (Fig. 6.5) armazena o elemento `CannonGameFragment` do aplicativo **Cannon Game**. Neste aplicativo, sobrescrevemos apenas o método `onCreate` de `Activity`, o qual infla a interface gráfica do usuário.

```
1 // MainActivity.java
2 // MainActivity exibe o elemento CannonGameFragment
3 package com.deitel.cannongame;
4
5 import android.app.Activity;
6 import android.os.Bundle;
7
8 public class MainActivity extends Activity
9 {
10    // chamado quando o aplicativo é ativado pela primeira vez
11    @Override
12    public void onCreate(Bundle savedInstanceState)
13    {
14        super.onCreate(savedInstanceState); // chama o método onCreate de super
15        setContentView(R.layout.activity_main); // infla o layout
16    }
17 } // fim da classe MainActivity
```

Figura 6.5 `MainActivity` exibe o elemento `CannonGameFragment`.

6.7 Subclasse CannonGameFragment de Fragment

A classe CannonGameFragment (Fig. 6.6) sobrescreve quatro métodos de Fragment:

- onCreateView (linhas 17 a 28) – Conforme você aprendeu na Seção 5.3.3, este método é chamado depois do método onCreate de um objeto Fragment para construir e retornar um objeto View contendo a interface gráfica do fragmento. As linhas 22 e 23 inflam a interface gráfica do usuário. A linha 26 obtém uma referência para o objeto CannonView de CannonGameFragment a fim de que possamos chamar seus métodos.
- onActivityCreated (linhas 31 a 38) – Este método é chamado depois que a atividade que armazena o fragmento é criada. A linha 37 chama o método setVolumeControlStream de Activity para permitir que o áudio do jogo seja controlado pelas teclas de volume do dispositivo.
- onPause (linhas 41 a 46) – Quando MainActivity é enviada para o *segundo plano* (e, assim, pausada), o método onPause de CannonGameFragment é executado. A linha 45 chama o método stopGame de CannonView (Seção 6.8.11) para interromper o loop do jogo.
- onDestroy (linhas 49 a 54) – Quando MainActivity é destruída, seu método onDestroy chama onDestroy de CannonGameFragment. A linha 46 chama o método releaseResources de CannonView (Seção 6.8.11) para liberar os recursos de som.

```

1 // CannonGameFragment.java
2 // CannonGameFragment cria e gerencia um componente CannonView
3 package com.deitel.cannongame;
4
5 import android.app.Fragment;
6 import android.media.AudioManager;
7 import android.os.Bundle;
8 import android.view.LayoutInflater;
9 import android.view.View;
10 import android.view.ViewGroup;
11
12 public class CannonGameFragment extends Fragment
13 {
14     private CannonView cannonView; // view personalizada para mostrar o jogo
15
16     // chamado quando a view do fragmento precisa ser criada
17     @Override
18     public View onCreateView(LayoutInflater inflater, ViewGroup container,
19         Bundle savedInstanceState)
20     {
21         super.onCreateView(inflater, container, savedInstanceState);
22         View view =
23             inflater.inflate(R.layout.fragment_game, container, false);
24
25         // obtém o componente CannonView
26         cannonView = (CannonView) view.findViewById(R.id.cannonView);
27         return view;
28     }
29
30     // configura o controle de volume quando a atividade é criada
31     @Override
32     public void onActivityCreated(Bundle savedInstanceState)
33     {

```

Figura 6.6 CannonGameFragment cria e gerencia um componente CannonView. (continua)

```
34     super.onCreate(savedInstanceState);
35
36     // permite que as teclas de volume ajustem o volume do jogo
37     getActivity().setVolumeControlStream(AudioManager.STREAM_MUSIC);
38 }
39
40 // quando MainActivity é pausada, CannonGameFragment termina o jogo
41 @Override
42 public void onPause()
43 {
44     super.onPause();
45     cannonView.stopGame(); // termina o jogo
46 }
47
48 // quando MainActivity é pausada, CannonGameFragment libera os recursos
49 @Override
50 public void onDestroy()
51 {
52     super.onDestroy();
53     cannonView.releaseResources();
54 }
55 } // fim da classe CannonGameFragment
```

Figura 6.6 CannonGameFragment cria e gerencia um componente CannonView.

6.8 Subclasse CannonView de View

A classe CannonView (Figs. 6.7 a 6.20) é uma subclasse personalizada de View que implementa a lógica do aplicativo Cannon Game e desenha objetos do jogo na tela.

6.8.1 As instruções package e import

A Figura 6.7 lista a instrução package e as instruções import da classe CannonView. A Seção 6.3 discute as novas classes e interfaces importantes utilizadas pela classe CannonView. Nós as realçamos na Figura 6.7.

```
1  // CannonView.java
2  // Exibe e controla o aplicativo Cannon Game
3  package com.deitel.cannongame;
4
5  import android.app.Activity;
6  import android.app.AlertDialog;
7  import android.app.Dialog;
8  import android.app.DialogFragment;
9  import android.content.Context;
10 import android.content.DialogInterface;
11 import android.graphics.Canvas;
12 import android.graphics.Color;
13 import android.graphics.Paint;
14 import android.graphics.Point;
15 import android.media.AudioManager;
16 import android.media.SoundPool;
17 import android.os.Bundle;
18 import android.util.AttributeSet;
19 import android.util.Log;
20 import android.util.SparseIntArray;
21 import android.view.MotionEvent;
22 import android.view.SurfaceHolder;
23 import android.view.SurfaceView;
24
```

Figura 6.7 Instruções package e import da classe CannonView.

6.8.2 Variáveis de instância e constantes

A Figura 6.8 lista o grande número de constantes e variáveis de instância da classe `CannonView`. A maioria é óbvia, mas vamos explicar cada uma delas à medida que as encontrarmos na discussão.

```

25 public class CannonView extends SurfaceView
26     implements SurfaceHolder.Callback
27 {
28     private static final String TAG = "CannonView"; // para registrar erros
29
30     private CannonThread cannonThread; // controla o loop do jogo
31     private Activity activity; // para exibir a caixa de diálogo Game Over na
        // thread da interface gráfica do usuário
32     private boolean dialogIsDisplayed = false;
33
34     // constantes para interação do jogo
35     public static final int TARGET_PIECES = 7; // seções no alvo
36     public static final int MISS_PENALTY = 2; // segundos subtraídos em caso de erro
37     public static final int HIT_REWARD = 3; // segundos adicionados em caso de acerto
38
39     // variáveis para o loop do jogo e controle de estatísticas
40     private boolean gameOver; // o jogo terminou?
41     private double timeLeft; // tempo restante em segundos
42     private int shotsFired; // tiros disparados pelo usuário
43     private double totalElapsedTime; // segundos decorridos
44
45     // variáveis para a barreira e para o alvo
46     private Line blocker; // pontos inicial e final da barreira
47     private int blockerDistance; // distância da barreira a partir da esquerda
48     private int blockerBeginning; // distância do topo da barreira até a parte superior
49     private int blockerEnd; // distância da parte inferior da barreira até o topo
50     private int initialBlockerVelocity; // multiplicador de velocidade inicial da barreira
51     private float blockerVelocity; // multiplicador de velocidade da barreira durante o jogo
52
53     private Line target; // pontos inicial e final do alvo
54     private int targetDistance; // distância do alvo a partir da esquerda
55     private int targetBeginning; // distância do alvo a partir do topo
56     private double pieceLength; // comprimento de uma parte do alvo
57     private int targetEnd; // distância da parte inferior do alvo a partir do topo
58     private int initialTargetVelocity; // multiplicador de velocidade inicial do alvo
59     private float targetVelocity; // multiplicador de velocidade do alvo
60
61     private int lineWidth; // largura do alvo e da barreira
62     private boolean[] hitStates; // cada parte do alvo foi atingida?
63     private int targetPiecesHit; // número de partes do alvo atingidas (até 7)
64
65     // variáveis para o canhão e para a bala
66     private Point cannonball; // canto superior esquerdo da imagem da bala
67     private int cannonballVelocityX; // velocidade x da bala
68     private int cannonballVelocityY; // velocidade y da bala
69     private boolean cannonballOnScreen; // se a bala está na tela ou não
70     private int cannonballRadius; // raio da bala
71     private int cannonballSpeed; // velocidade da bala
72     private int cannonBaseRadius; // raio da base do canhão
73     private int cannonLength; // comprimento do cano do canhão
74     private Point barrelEnd; // o ponto extremo do cano do canhão
75     private int screenWidth;
76     private int screenHeight;
77

```

Figura 6.8 Campos da classe `CannonView`. (continua)


```

78 // constantes e variáveis para gerenciar sons
79 private static final int TARGET_SOUND_ID = 0;
80 private static final int CANNON_SOUND_ID = 1;
81 private static final int BLOCKER_SOUND_ID = 2;
82 private SoundPool soundPool; // reproduz os efeitos sonoros
83 private SparseIntArray soundMap; // mapeia identificadores em SoundPool
84
85 // variáveis Paint utilizadas ao desenhar cada item na tela
86 private Paint textPaint; // objeto Paint usado para desenhar texto
87 private Paint cannonballPaint; // objeto Paint usado para desenhar a bala de canhão
88 private Paint cannonPaint; // objeto Paint usado para desenhar o canhão
89 private Paint blockerPaint; // objeto Paint usado para desenhar a barreira
90 private Paint targetPaint; // objeto Paint usado para desenhar o alvo
91 private Paint backgroundPaint; // objeto Paint usado para limpar a área de desenho
92

```

Figura 6.8 Campos da classe CannonView.

6.8.3 Construtor

A Figura 6.9 mostra o construtor da classe CannonView. Quando um objeto View é inflado, seu construtor é chamado com um objeto Context e um AttributeSet como argumentos. O objeto Context é a atividade que exibe o elemento CannonGameFragment que contém o componente CannonView, e AttributeSet (pacote android.util) contém os valores de atributo de CannonView configurados no documento XML do layout. Esses argumentos são passados para o construtor da superclasse (linha 96) para garantir que o objeto View personalizado seja corretamente configurado com os valores de quaisquer atributos padrão de View especificados no código XML. A linha 97 armazena uma referência para o objeto MainActivity a fim de que possamos usá-lo ao final de um jogo para exibir um componente AlertDialog a partir da thread da interface gráfica da atividade.

```

93 // construtor public
94 public CannonView(Context context, AttributeSet attrs)
95 {
96     super(context, attrs); // chama o construtor da superclasse
97     activity = (Activity) context; // armazena referência para MainActivity
98
99     // registra o receptor SurfaceHolder.Callback
100     getHolder().addCallback(this);
101
102     // inicializa Lines e Point representando itens do jogo
103     blocker = new Line(); // cria a barreira como um objeto Line
104     target = new Line(); // cria o alvo como um objeto Line
105     cannonball = new Point(); // cria a bala de canhão como um objeto Point
106
107     // inicializa hitStates como um array booleano
108     hitStates = new boolean[TARGET_PIECES];
109
110     // inicializa SoundPool para reproduzir os três efeitos sonoros do aplicativo
111     soundPool = new SoundPool(1, AudioManager.STREAM_MUSIC, 0);
112
113     // cria objeto Map de sons e carrega os sons previamente
114     soundMap = new SparseIntArray(3); // cria novo objeto SparseIntArray
115     soundMap.put(TARGET_SOUND_ID,
116         soundPool.load(context, R.raw.target_hit, 1));
117     soundMap.put(CANNON_SOUND_ID,
118         soundPool.load(context, R.raw.cannon_fire, 1));
119     soundMap.put(BLOCKER_SOUND_ID,

```

Figura 6.9 Construtor de CannonView.

```

120     soundPool.load(context, R.raw.blocker_hit, 1));
121
122     // constrói objetos Paint para desenhar o texto, a bala, o canhão,
123     // a barreira e o alvo; eles são configurados no método onSizeChanged
124     textPaint = new Paint();
125     cannonPaint = new Paint();
126     cannonballPaint = new Paint();
127     blockerPaint = new Paint();
128     targetPaint = new Paint();
129     backgroundPaint = new Paint();
130 } // fim do construtor de CannonView
131

```

Figura 6.9 Construtor de CannonView.

Registrando o receptor *SurfaceHolder.Callback*

A linha 100 registra *this* (isto é, o elemento *CannonView*) como o objeto que implementa *SurfaceHolder.Callback* para receber as chamadas de método que indicam quando o objeto *SurfaceView* é *criado, atualizado e destruído*. O método herdado *getHolder* de *SurfaceView* retorna o objeto *SurfaceHolder* correspondente para gerenciar *SurfaceView*, e o método *addCallback* de *SurfaceHolder* armazena o objeto que implementa a interface *SurfaceHolder.Callback*.

Criando a barreira, o alvo e a bala de canhão

As linhas 103 a 105 criam a barreira e o alvo como objetos *Line* e a bala de canhão como um objeto *Point*. Em seguida, criamos o array booleano *hitStates* para controlar quais das sete partes do alvo foram atingidas (e que, portanto, não devem ser desenhadas).

Configurando o componente *SoundPool* e carregando os sons

As linhas 111 a 120 configuram os sons que usamos no aplicativo. Primeiramente, criamos o objeto *SoundPool* utilizado para carregar e reproduzir os efeitos sonoros do aplicativo. O primeiro argumento do construtor representa o número máximo de fluxos de som simultâneos que podem ser reproduzidos. Reproduzimos apenas um som por vez; portanto, passamos 1. O segundo argumento especifica qual fluxo de áudio vai ser usado para reproduzir os sons. Existem sete fluxos de som, identificados pelas constantes na classe *AudioManager*, mas a documentação da classe *SoundPool* recomenda usar o fluxo para tocar música (*AudioManager.STREAM_MUSIC*) para som em jogos. O último argumento representa a qualidade do som, mas a documentação indica que esse valor não é usado atualmente e que 0 deve ser especificado como valor padrão.

A linha 114 cria um elemento *SparseIntArray* (*soundMap*) que mapeia chaves inteiras a valores inteiros. *SparseIntArray* é semelhante – porém, mais eficiente – a um elemento *HashMap<Integer, Integer>* para números pequenos de pares chave-valor. Neste caso, mapeamos as chaves de som (definidas na Fig. 6.8, linhas 79 a 81) para os identificadores dos sons carregados, os quais são representados pelos valores de retorno do método *load* de *SoundPool* (chamado na Fig. 6.9, linhas 116, 118 e 120). Cada identificador de som pode ser usado para *reproduzir* um som (e depois para devolver seus recursos para o sistema). O método *load* de *SoundPool* recebe três argumentos – o objeto *Context* do aplicativo, um identificador de recurso que representa o arquivo de som a ser carregado e a prioridade do som. De acordo com a documentação desse método, o último argumento não é usado atualmente e deve ser especificado como 1.

Criando os objetos Paint usados para desenhar os elementos do jogo

As linhas 124 a 129 criam os objetos Paint utilizados ao se desenhar os elementos do jogo. Configuramos isso no método `onSizeChanged` (Seção 6.8.4), pois algumas das configurações de Paint dependem de mudanças na escala dos elementos do jogo com base no tamanho da tela do dispositivo.

6.8.4 Sobrescrevendo o método `onSizeChanged` de View

A Figura 6.10 sobrescreve o **método `onSizeChanged`** da classe View, o qual é chamado quando o tamanho da View muda, inclusive quando o objeto View é adicionado à hierarquia de Views, quando o layout é inflado. Este aplicativo sempre aparece no *modo retrato*; portanto, `onSizeChanged` é chamado somente uma vez, quando o método `onCreate` da atividade infla a interface gráfica do usuário. O método recebe a largura e altura novas do objeto View e a largura e altura antigas – quando esse método é chamado pela primeira vez, a largura e altura antigas são 0. Os cálculos efetuados aqui *mudam a escala* dos elementos do jogo na tela com base na largura e na altura em pixels do dispositivo. Chegamos a nossos fatores de escala por meio de tentativa e erro, escolhendo valores que faziam os elementos do jogo ter a melhor aparência na tela. As linhas 170 a 175 configuram os objetos Paint utilizados ao especificar as características de desenho dos elementos do jogo. Após os cálculos, a linha 177 chama o método `newGame` (Fig. 6.11).

```

132 // chamado por surfaceChanged quando o tamanho do componente SurfaceView
133 // muda, como quando ele é adicionado à hierarquia de Views
134 @Override
135 protected void onSizeChanged(int w, int h, int oldw, int oldh)
136 {
137     super.onSizeChanged(w, h, oldw, oldh);
138
139     screenWidth = w; // armazena a largura de CannonView
140     screenHeight = h; // armazena a altura de CannonView
141     cannonBaseRadius = h / 18; // o raio da base do canhão tem 1/18 da altura da tela
142     cannonLength = w / 8; // o comprimento do canhão tem 1/8 da largura da tela
143
144     cannonballRadius = w / 36; // o raio da bala tem 1/36 da largura da tela
145     cannonballSpeed = w * 3 / 2; // multiplicador de velocidade da bala
146
147     lineWidth = w / 24; // o alvo e a barreira têm 1/24 da largura da tela
148
149     // configura variáveis de instância relacionadas à barreira
150     blockerDistance = w * 5 / 8; // a barreira tem 5/8 da largura da tela a partir
    // da esquerda
151     blockerBeginning = h / 8; // a distância a partir do topo é de 1/8 da altura da tela
152     blockerEnd = h * 3 / 8; // a distância a partir do topo é de 3/8 da altura da tela
153     initialBlockerVelocity = h / 2; // multiplicador de velocidade inicial da barreira
154     blocker.start = new Point(blockerDistance, blockerBeginning);
155     blocker.end = new Point(blockerDistance, blockerEnd);
156
157     // configura variáveis de instância relacionadas ao alvo
158     targetDistance = w * 7 / 8; // o alvo tem 7/8 da largura da tela a partir da esquerda
159     targetBeginning = h / 8; // a distância a partir do topo é de 1/8 da altura da tela
160     targetEnd = h * 7 / 8; // a distância a partir do topo é de 7/8 da altura da tela
161     pieceLength = (targetEnd - targetBeginning) / TARGET_PIECES;
162     initialTargetVelocity = -h / 4; // multiplicador de velocidade inicial do alvo
163     target.start = new Point(targetDistance, targetBeginning);

```

Figura 6.10 Método `onSizeChanged` sobrescrito.

```

164     target.end = new Point(targetDistance, targetEnd);
165
166     // o ponto extremo do cano do canhão aponta horizontalmente no início
167     barrelEnd = new Point(cannonLength, h / 2);
168
169     // configura objetos Paint para desenhar os elementos do jogo
170     textPaint.setTextSize(w / 20); // o tamanho do texto tem 1/20 da largura da tela
171     textPaint.setAntiAlias(true); // suaviza o texto
172     cannonPaint.setStrokeWidth(lineWidth * 1.5f); // configura a espessura da linha
173     blockerPaint.setStrokeWidth(lineWidth); // configura a espessura da linha
174     targetPaint.setStrokeWidth(lineWidth); // configura a espessura da linha
175     backgroundPaint.setColor(Color.WHITE); // configura a cor de fundo
176
177     newGame(); // prepara e inicia um novo jogo
178 } // fim do método onSizeChanged
179

```

Figura 6.10 Método onSizeChanged sobrescrito.

6.8.5 Método newGame

O método newGame (Fig. 6.11) reinicia as variáveis de instância utilizadas para controlar o jogo. Se a variável gameOver for true – o que ocorre somente *após* o primeiro jogo terminar –, a linha 203 zera gameOver e as linhas 204 e 205 criam um novo objeto CannonThread e o iniciam para começar o *loop* que controla o jogo. Você vai aprender mais sobre isso na Seção 6.8.14.

```

180 // reinicia todos os elementos de tela e inicia um novo jogo
181 public void newGame()
182 {
183     // configura cada elemento de hitStates como false -- restaura partes do alvo
184     for (int i = 0; i < TARGET_PIECES; i++)
185         hitStates[i] = false;
186
187     targetPiecesHit = 0; // nenhuma parte do alvo foi atingida
188     blockerVelocity = initialBlockerVelocity; // configura a velocidade inicial
189     targetVelocity = initialTargetVelocity; // configura a velocidade inicial
190     timeLeft = 10; // inicia a contagem regressiva em 10 segundos
191     cannonballOnScreen = false; // a bala de canhão não está na tela
192     shotsFired = 0; // configura o número inicial de tiros disparados
193     totalElapsedTime = 0.0; // configura o tempo decorrido como zero
194
195     // configura os objetos Point inicial e final da barreira e do alvo
196     blocker.start.set(blockerDistance, blockerBeginning);
197     blocker.end.set(blockerDistance, blockerEnd);
198     target.start.set(targetDistance, targetBeginning);
199     target.end.set(targetDistance, targetEnd);
200
201     if (gameOver) // iniciando um novo jogo depois que o último terminou
202     {
203         gameOver = false; // o jogo não terminou
204         cannonThread = new CannonThread(getHolder()); // cria thread
205         cannonThread.start(); // inicia a thread do loop do jogo
206     } // fim do if
207 } // fim do método newGame
208

```

Figura 6.11 Método newGame de CannonView.

6.8.6 Método updatePositions

O método `updatePositions` (Fig. 6.12) é chamado pelo método `run` de `CannonThread` (Seção 6.8.14) para atualizar as posições dos elementos na tela e para fazer a *detecção de colisão* simples. Os novos locais dos elementos do jogo são calculados com base no tempo decorrido, em milissegundos, entre o quadro anterior e o quadro atual da animação. Isso permite que o jogo atualize a quantidade pela qual cada elemento se move com base na *taxa de atualização* do dispositivo. Discutiremos isso com mais detalhes quando abordarmos os loops do jogo na Seção 6.8.14.

```

209 // chamado repetidamente por CannonThread para atualizar os elementos do jogo
210 private void updatePositions(double elapsedTimeMS)
211 {
212     double interval = elapsedTimeMS / 1000.0; // converte em segundos
213
214     if (cannonballOnScreen) // se um tiro foi disparado no momento
215     {
216         // atualiza a posição da bala de canhão
217         cannonball.x += interval * cannonballVelocityX;
218         cannonball.y += interval * cannonballVelocityY;
219
220         // verifica se houve colisão com a barreira
221         if (cannonball.x + cannonballRadius > blockerDistance &&
222             cannonball.x - cannonballRadius < blockerDistance &&
223             cannonball.y + cannonballRadius > blocker.start.y &&
224             cannonball.y - cannonballRadius < blocker.end.y)
225         {
226             cannonballVelocityX *= -1; // direção inversa da bala de canhão
227             timeLeft -= MISS_PENALTY; // penaliza o usuário
228
229             // reproduz o som da barreira
230             soundPool.play(soundMap.get(BLOCKER_SOUND_ID), 1, 1, 1, 0, 1f);
231         }
232         // verifica se houve colisões com as paredes esquerda e direita
233         else if (cannonball.x + cannonballRadius > screenWidth ||
234                 cannonball.x - cannonballRadius < 0)
235         {
236             cannonballOnScreen = false; // remove a bala de canhão da tela
237         }
238         // verifica se houve colisões com as paredes superior e inferior
239         else if (cannonball.y + cannonballRadius > screenHeight ||
240                 cannonball.y - cannonballRadius < 0)
241         {
242             cannonballOnScreen = false; // remove a bala de canhão da tela
243         }
244         // verifica se houve colisão da bala com o alvo
245         else if (cannonball.x + cannonballRadius > targetDistance &&
246                 cannonball.x - cannonballRadius < targetDistance &&
247                 cannonball.y + cannonballRadius > target.start.y &&
248                 cannonball.y - cannonballRadius < target.end.y)
249         {
250             // determina o número da seção do alvo (0 é a parte superior)
251             int section =
252                 (int) ((cannonball.y - target.start.y) / pieceLength);
253
254             // verifica se a parte ainda não foi atingida
255             if ((section >= 0 && section < TARGET_PIECES) &&

```

Figura 6.12 Método `updatePositions` de `CannonView`.

```

256         !hitStates[section])
257     {
258         hitStates[section] = true; // a seção foi atingida
259         cannonballOnScreen = false; // remove a bala de canhão
260         timeLeft += HIT_REWARD; // acrescenta recompensa ao tempo restante
261
262         // reproduz o som de alvo atingido
263         soundPool.play(soundMap.get(TARGET_SOUND_ID), 1,
264             1, 1, 0, 1f);
265
266         // se todas as partes foram atingidas
267         if (++targetPiecesHit == TARGET_PIECES)
268         {
269             cannonThread.setRunning(false); // termina a thread
270             showGameOverDialog(R.string.win); // mostra caixa de diálogo
271                                     // de vitória
272             gameOver = true;
273         }
274     }
275 }
276
277 // atualiza a posição da barreira
278 double blockerUpdate = interval * blockerVelocity;
279 blocker.start.y += blockerUpdate;
280 blocker.end.y += blockerUpdate;
281
282 // atualiza a posição do alvo
283 double targetUpdate = interval * targetVelocity;
284 target.start.y += targetUpdate;
285 target.end.y += targetUpdate;
286
287 // se a barreira atingiu a parte superior ou inferior, inverte a direção
288 if (blocker.start.y < 0 || blocker.end.y > screenHeight)
289     blockerVelocity *= -1;
290
291 // se o alvo atingiu a parte superior ou inferior, inverte a direção
292 if (target.start.y < 0 || target.end.y > screenHeight)
293     targetVelocity *= -1;
294
295 timeLeft -= interval; // subtrai do tempo restante
296
297 // se o cronômetro foi zerado
298 if (timeLeft <= 0.0)
299 {
300     timeLeft = 0.0;
301     gameOver = true; // o jogo terminou
302     cannonThread.setRunning(false); // termina a thread
303     showGameOverDialog(R.string.lose); // mostra caixa de diálogo de derrota
304 }
305 } // fim do método updatePositions
306

```

Figura 6.12 Método updatePositions de CannonView.

Tempo decorrido desde o último quadro de animação

A linha 212 converte o tempo decorrido desde o último quadro da animação, de milissegundos para segundos. Esse valor é usado para modificar as posições de vários elementos do jogo.

Verificando se houve colisões com a barreira

A linha 214 verifica se a bala de canhão está na tela. Se estiver, atualizamos sua posição adicionando a distância que ela deve ter percorrido desde o último evento do cronômetro. Isso é calculado multiplicando sua velocidade pela quantidade de tempo decorrido (linhas 217 e 218). As linhas 221 a 224 verificam se a bala de canhão *colidiu* com a barreira. Fazemos *detecção de colisão* simples, com base no limite retangular da bala de canhão. Existem quatro condições que devem ser satisfeitas se a bala de canhão estiver em contato com a barreira:

- A coordenada x da bala de canhão mais o raio da bala devem ser maiores que a distância da barreira a partir da margem esquerda da tela (`blockerDistance`) (linha 221). Isso significa que a bala de canhão atingiu a distância da barreira a partir da margem esquerda da tela.
- A coordenada x da bala de canhão menos o raio da bala também deve ser menor que a distância da barreira a partir da margem esquerda da tela (linha 222). Isso garante que a bala de canhão ainda não passou pela barreira.
- Parte da bala de canhão deve estar mais baixa do que a parte superior da barreira (linha 223).
- Parte da bala de canhão deve estar mais alta do que a parte inferior da barreira (linha 224).

Se todas essas condições forem satisfeitas, *invertemos* a direção da bala de canhão na tela (linha 226), *penalizamos* o usuário *subtraindo* `MISS_PENALTY` de `timeLeft` e, então, chamamos o **método** `play` de `SoundPool` para reproduzir o som de golpe na barreira – `BLOCKER_SOUND_ID` é usado como chave de `soundMap` para localizar o identificador de som em `SoundPool`.

Verificando se a bala de canhão saiu da tela

Removemos a bala de canhão se ela atingir qualquer uma das margens da tela. As linhas 233 a 237 testam se a bala de canhão *colidiu* com a parede esquerda ou direita e, caso tenha colidido, removem a bala de canhão da tela. As linhas 239 a 243 removem a bala de canhão se ela colidir com a parte superior ou inferior da tela.

Verificando se houve colisões com o alvo

Em seguida, verificamos se a bala de canhão atingiu o alvo (linhas 245 a 248). As condições são semelhantes às usadas para determinar se a bala de canhão colidiu com a barreira. Se a bala atingiu o alvo, as linhas 251 e 252 determinam a *seção* atingida – dividindo a distância entre a bala de canhão e a parte inferior do alvo pelo comprimento de uma parte. Essa expressão é avaliada como 0 para a seção superior e como 6 para a inferior. Verificamos se essa seção foi atingida anteriormente usando o array `hitStates` (linha 256). Caso não tenha sido, configuramos o elemento `hitStates` correspondente como `true` e removemos a bala de canhão da tela. Então, somamos `HIT_REWARD` a `timeLeft`, aumentando o tempo restante do jogo, e reproduzimos o som de golpe no alvo (`TARGET_SOUND_ID`). Incrementamos `targetPiecesHit` e, em seguida, determinamos se ele é igual a `TARGET_PIECES` (linha 267). Se for, o jogo terminou; portanto, terminamos `CannonThread` chamando seu método `setRunning` com o argumento `false`, chamamos o método `showGameOverDialog` com o identificador do recurso de `String` que representa a mensagem de vitória e configuramos `gameOver` como `true`.

Atualizando as posições da barreira e do alvo

Agora que todas as colisões possíveis da bala de canhão foram verificadas, as posições da barreira e do alvo devem ser atualizadas. As linhas 278 a 280 mudam a posição da barreira multiplicando `blockerVelocity` pela quantidade de tempo decorrido desde a última atualização e somando esse valor às coordenadas x e y atuais. As linhas 283 a 285 fazem o mesmo para o alvo. Se a barreira colidiu com a parede superior ou inferior, sua direção é *invertida*, multiplicando-se sua velocidade por -1 (linhas 288 e 289). As linhas 292 e 293 fazem a mesma verificação e ajuste para o comprimento total do alvo, incluindo quaisquer seções que já foram destruídas.

Atualizando o tempo restante e determinando se o tempo esgotou

Diminuímos `timeLeft` do tempo decorrido desde o quadro de animação anterior (linha 295). Se `timeLeft` chegar a zero, o jogo terminou – configuramos `timeLeft` como 0.0 para o caso de ser negativo (caso contrário, às vezes mostraríamos um tempo final negativo na tela). Então, configuramos `gameOver` como `true`, terminamos `CannonThread` chamando seu método `setRunning` com o argumento `false` e chamamos o método `showGameOverDialog` com o identificador do recurso de `String` que representa a mensagem de derrota.

6.8.7 Método `fireCannonball`

Quando o usuário *toca* na tela, o método `onTouchEvent` (Seção 6.8.13) chama `fireCannonball` (Fig. 6.13). Se já existe uma bala de canhão na tela, o método *retorna imediatamente*. A linha 313 chama `alignCannon` para mirar o canhão no *ponto do toque* e obter o ângulo do canhão. As linhas 316 e 317 “carregam o canhão” (isto é, posicionam a bala dentro do canhão). Em seguida, as linhas 320 e 323 calculam os componentes horizontal e vertical da velocidade da bala. Então, configuramos `cannonballOnScreen` como `true` para que a bala seja desenhada pelo método `drawGameElements` (Fig. 6.15) e incrementamos `shotsFired`. Por fim, reproduzimos o som de disparo do canhão (representado por `CANNON_SOUND_ID`).

```

307 // dispara uma bala de canhão
308 public void fireCannonball(MotionEvent event)
309 {
310     if (cannonballOnScreen) // se uma bala já está na tela
311         return; // nada faz
312
313     double angle = alignCannon(event); // obtém o ângulo do cano do canhão
314
315     // move a bala para dentro do canhão
316     cannonball.x = cannonballRadius; // alinha a coordenada x com o canhão
317     cannonball.y = screenHeight / 2; // centraliza a bala verticalmente
318
319     // obtém o componente x da velocidade total
320     cannonballVelocityX = (int) (cannonballSpeed * Math.sin(angle));
321
322     // obtém o componente y da velocidade total
323     cannonballVelocityY = (int) (-cannonballSpeed * Math.cos(angle));
324     cannonballOnScreen = true; // a bala de canhão está na tela
325     ++shotsFired; // incrementa shotsFired
326
327     // reproduz o som de canhão disparado
328     soundPool.play(soundMap.get(CANNON_SOUND_ID), 1, 1, 1, 0, 1f);
329 } // fim do método fireCannonball
330

```

Figura 6.13 Método `fireCannonball` de `CannonView`.

6.8.8 Método alignCannon

O método alignCannon (Fig. 6.14) mira o canhão para o ponto onde o usuário tocou na tela. A linha 335 obtém as coordenadas x e y do *toque* a partir do argumento MotionEvent. Calculamos a distância vertical do toque a partir do centro da tela. Se ela não for zero, calculamos o ângulo do cano do canhão a partir da horizontal (linha 345). Se o toque ocorrer na metade inferior da tela, ajustamos o ângulo com Math.PI (linha 349). Então, usamos cannonLength e angle para determinar os valores de coordenada x e y do ponto extremo do cano do canhão – isso é usado para desenhar uma linha do centro da base do canhão na margem esquerda da tela até o ponto extremo do cano do canhão.

```

331 // alinha o canhão em resposta a um toque do usuário
332 public double alignCannon(MotionEvent event)
333 {
334     // obtém o local do toque nessa view de exibição
335     Point touchPoint = new Point((int) event.getX(), (int) event.getY());
336
337     // calcula a distância do toque a partir do centro da tela
338     // no eixo y
339     double centerMinusY = (screenHeight / 2 - touchPoint.y);
340
341     double angle = 0; // inicializa o ângulo com 0
342
343     // calcula o ângulo do cano em relação à horizontal
344     if (centerMinusY != 0) // evita divisão por 0
345         angle = Math.atan((double) touchPoint.x / centerMinusY);
346
347     // se o toque foi dado na metade inferior da tela
348     if (touchPoint.y > screenHeight / 2)
349         angle += Math.PI; // ajusta o ângulo
350
351     // calcula o ponto extremo do cano do canhão
352     barrelEnd.x = (int) (cannonLength * Math.sin(angle));
353     barrelEnd.y =
354         (int) (-cannonLength * Math.cos(angle) + screenHeight / 2);
355
356     return angle; // retorna o ângulo calculado
357 } // fim do método alignCannon
358

```

Figura 6.14 Método alignCannon de CannonView.

6.8.9 Método drawGameElements

O método drawGameElements (Fig. 6.15) desenha o *canhão*, a *bala*, a *barreira* e o *alvo* no componente SurfaceView usando o elemento Canvas que CannonThread (Seção 6.8.14) obtém do objeto SurfaceHolder de SurfaceView.

```

359 // desenha o jogo no objeto Canvas dado
360 public void drawGameElements(Canvas canvas)
361 {
362     // limpa o plano de fundo
363     canvas.drawRect(0, 0, canvas.getWidth(), canvas.getHeight(),
364         backgroundPaint);
365

```

Figura 6.15 Método drawGameElements de CannonView.

```

366 // exibe o tempo restante
367 canvas.drawText(getResources().getString(
368     R.string.time_remaining_format, timeLeft), 30, 50, textPaint);
369
370 // se uma bala de canhão está na tela, a desenha
371 if (cannonballOnScreen)
372     canvas.drawCircle(cannonball.x, cannonball.y, cannonballRadius,
373         cannonballPaint);
374
375 // desenha o cano do canhão
376 canvas.drawLine(0, screenHeight / 2, barrelEnd.x, barrelEnd.y,
377     cannonPaint);
378
379 // desenha a base do canhão
380 canvas.drawCircle(0, (int) screenHeight / 2,
381     (int) cannonBaseRadius, cannonPaint);
382
383 // desenha a barreira
384 canvas.drawLine(blocker.start.x, blocker.start.y, blocker.end.x,
385     blocker.end.y, blockerPaint);
386
387 Point currentPoint = new Point(); // início da seção do alvo atual
388
389 // inicializa currentPoint com o ponto inicial do alvo
390 currentPoint.x = target.start.x;
391 currentPoint.y = target.start.y;
392
393 // desenha o alvo
394 for (int i = 0; i < TARGET_PIECES; i++)
395 {
396     // se essa parte do alvo não foi atingida, a desenha
397     if (!hitStates[i])
398     {
399         // alterna o colorido das partes
400         if (i % 2 != 0)
401             targetPaint.setColor(Color.BLUE);
402         else
403             targetPaint.setColor(Color.YELLOW);
404
405         canvas.drawLine(currentPoint.x, currentPoint.y, target.end.x,
406             (int) (currentPoint.y + pieceLength), targetPaint);
407     }
408
409     // move currentPoint para o início da próxima parte
410     currentPoint.y += pieceLength;
411 }
412 } // fim do método drawGameElements
413

```

Figura 6.15 Método drawGameElements de CannonView.

Limpando o elemento Canvas com o método drawRect

Primeiramente, chamamos o **método drawRect** de Canvas (linhas 363 e 364) para limpar o componente Canvas a fim de que todos os elementos do jogo possam aparecer em suas novas posições. O método recebe como argumentos as coordenadas *x-y* do canto superior esquerdo do retângulo, a largura e altura do retângulo e o objeto Paint que especifica as características do desenho – lembre-se de que backgroundPaint configura a cor do desenho como branca.

Exibindo o tempo restante com o método `drawText` de Canvas

Em seguida, chamamos o método `drawText` de Canvas (linhas 367 e 368) para exibir o tempo restante no jogo. Passamos como argumentos para o objeto String a ser exibido as coordenadas x e y nas quais ele vai aparecer e o objeto `textPaint` (configurado nas linhas 170 e 171) para descrever como o texto deve ser apresentado (isto é, o tamanho da fonte, a cor e outros atributos do texto).

Desenhando a bala de canhão com o método `drawCircle` de Canvas

Se a bala de canhão estiver na tela, as linhas 372 e 373 usam o método `drawCircle` de Canvas para desenhá-la em sua posição atual. Os dois primeiros argumentos representam as coordenadas do *centro* do círculo. O terceiro argumento é o *raio* do círculo. O último argumento é o objeto `Paint` que especifica as características de desenho do círculo.

Desenhando o cano do canhão, a barreira e o alvo com o método `drawLine` de Canvas

Usamos o método `drawLine` de Canvas para exibir o *cano* do canhão (linhas 376 e 377), a *barreira* (linhas 384 e 385) e as *partes do alvo* (linhas 405 e 406). Esse método recebe cinco parâmetros – os quatro primeiros representam as coordenadas x - y do início e do fim da linha, e o último é o objeto `Paint` que especifica as características da linha, como sua espessura.

Desenhando a base do canhão com o método `drawCircle` de Canvas

As linhas 380 e 381 usam o método `drawCircle` de Canvas para desenhar a base semi-circular do canhão, desenhando um círculo centralizado na margem esquerda da tela – como um círculo é exibido com base em seu ponto central, metade dele é desenhada fora do lado esquerdo do componente `SurfaceView`.

Desenhando as seções do alvo com o método `drawLine` de Canvas

As linhas 390 a 411 desenhavam as seções do alvo. Iteramos pelas seções, desenhando cada uma na cor correta – azul para as partes de numeração ímpar, amarelo para as outras. São exibidas somente as seções que não foram atingidas.

6.8.10 Método `showGameOverDialog`

Quando o jogo termina, o método `showGameOverDialog` (Fig. 6.16) exibe um elemento `DialogFragment` (usando as técnicas que você aprendeu na Seção 5.6.9) contendo um componente `AlertDialog` que indica se o jogador ganhou ou perdeu, o número de tiros disparados e o tempo total decorrido. A chamada do método `setPositiveButton` (linhas 433 a 444) cria um botão de reinício para começar um novo jogo.

```

414 // exibe um componente AlertDialog quando o jogo termina
415 private void showGameOverDialog(final int messageId)
416 {
417     // DialogFragment para exibir estatísticas do jogo e começar um novo teste
418     final DialogFragment gameResult =
419         new DialogFragment()
420     {
421         // cria um componente AlertDialog e o retorna
422         @Override
423         public Dialog onCreateDialog(Bundle bundle)

```

Figura 6.16 Método `showGameOverDialog` de `CannonView`.

```

424         {
425             // cria caixa de diálogo exibindo recurso String pelo messageId
426             AlertDialog.Builder builder =
427                 new AlertDialog.Builder(getActivity());
428             builder.setTitle(getResources().getString(messageId));
429
430             // exibe o número de tiros disparados e o tempo total decorrido
431             builder.setMessage(getResources().getString(
432                 R.string.results_format, shotsFired, totalElapsedTime));
433             builder.setPositiveButton(R.string.reset_game,
434                 new DialogInterface.OnClickListener()
435                 {
436                     // chamado quando o componente Button "Reset Game" é pressionado
437                     @Override
438                     public void onClick(DialogInterface dialog, int which)
439                     {
440                         dialogIsDisplayed = false;
441                         newGame(); // prepara e inicia um novo jogo
442                     }
443                 } // fim da classe interna anônima
444             ); // fim da chamada a setPositiveButton
445
446             return builder.create(); // retorna o componente AlertDialog
447         } // fim do método onCreateDialog
448     }; // fim da classe interna anônima DialogFragment
449
450     // em uma thread de interface gráfica do usuário, usa FragmentManager para
451     // exibir o componente DialogFragment
452     activity.runOnUiThread(
453         new Runnable() {
454             public void run()
455             {
456                 dialogIsDisplayed = true;
457                 gameResult.setCancellable(false); // caixa de diálogo modal
458                 gameResult.show(activity.getFragmentManager(), "results" );
459             }
460         } // fim de Runnable
461     ); // fim da chamada a runOnUiThread
462 } // fim do método showGameOverDialog

```

Figura 6.16 Método showGameOverDialog de CannonView.

O método `onClick` do receptor do botão indica que a caixa de diálogo não está mais aparecendo e chama `newGame` para configurar e iniciar um novo jogo. Uma caixa de diálogo deve ser exibida a partir da thread da interface gráfica do usuário, de modo que as linhas 451 a 460 chamam o método `runOnUiThread` de Activity para especificar um objeto `Runnable`, que deve ser executado na thread da interface assim que possível. O argumento é um objeto de uma classe interna anônima que implementa `Runnable`. O método `run` de `Runnable` indica que a caixa de diálogo está sendo exibida e, então, a exibe.

6.8.11 Métodos `stopGame` e `releaseResources`

Os métodos `onPause` e `onDestroy` da classe `CannonGameFragment` (Seção 6.7) chamam os métodos `stopGame` e `releaseResources` da classe `CannonView`, respectivamente (Fig. 6.17). O método `stopGame` (linhas 464 a 468) é chamado a partir do objeto Activity principal para interromper o jogo quando o método `onPause` de Activity é chamado – por simplicidade, não armazenamos o estado do jogo neste exemplo. O método `releaseResources`

(linhas 471 a 475) chama o método **release** de `SoundPool` para liberar os recursos associados ao objeto `SoundPool`.

```

463 // interrompe o jogo; chamado pelo método onPause de CannonGameFragment
464 public void stopGame()
465 {
466     if (cannonThread != null)
467         cannonThread.setRunning(false); // diz à thread para terminar
468 }
469
470 // libera recursos; chamado pelo método onDestroy de CannonGame
471 public void releaseResources()
472 {
473     soundPool.release(); // libera todos os recursos usados por SoundPool
474     soundPool = null;
475 }
476

```

Figura 6.17 Métodos `stopGame` e `releaseResources` de `CannonView`.

6.8.12 Implementando os métodos de `SurfaceHolder.Callback`

A Figura 6.18 implementa os métodos **surfaceChanged**, **surfaceCreated** e **surfaceDestroyed** da interface `SurfaceHolder.Callback`. O método `surfaceChanged` tem um corpo vazio neste aplicativo, pois este *sempre* é exibido na *orientação retrato*. Esse método é chamado quando o tamanho ou a orientação da `SurfaceView` muda, e normalmente seria usado para exibir os elementos gráficos outra vez, com base nessas alterações. O método `surfaceCreated` (linhas 485 a 494) é chamado quando o objeto `SurfaceView` é criado – por exemplo, quando o aplicativo é carregado pela primeira vez ou quando é retomado do segundo plano. Usamos `surfaceCreated` para criar e iniciar a thread `CannonThread` a fim de iniciar o loop de jogo. O método `surfaceDestroyed` (linhas 497 a 515) é chamado quando o objeto `SurfaceView` é destruído – por exemplo, quando o aplicativo termina. Usamos o método para garantir que `CannonThread` termine corretamente. Primeiramente, a linha 502 chama o método `setRunning` de `CannonThread` com `false` como argumento, para indicar que a thread deve *parar*; então, as linhas 504 a 515 esperam que a thread *termine*. Isso garante que não seja feita qualquer tentativa de desenhar na `SurfaceView` uma vez que `surfaceDestroyed` termine de ser executado.

```

477 // chamado quando o tamanho da superfície muda
478 @Override
479 public void surfaceChanged(SurfaceHolder holder, int format,
480     int width, int height)
481 {
482 }
483
484 // chamado quando a superfície é criada
485 @Override
486 public void surfaceCreated(SurfaceHolder holder)
487 {
488     if (!dialogIsDisplayed)
489     {
490         cannonThread = new CannonThread(holder); // cria a thread

```

Figura 6.18 Implementando os métodos de `SurfaceHolder.Callback`.

```

491         cannonThread.setRunning(true); // começa a executar o jogo
492         cannonThread.start(); // inicia a thread do loop do jogo
493     }
494 }
495
496 // chamado quando a superfície é destruída
497 @Override
498 public void surfaceDestroyed(SurfaceHolder holder)
499 {
500     // garante que essa thread termine corretamente
501     boolean retry = true;
502     cannonThread.setRunning(false); // termina cannonThread
503
504     while (retry)
505     {
506         try
507         {
508             cannonThread.join(); // espera cannonThread terminar
509             retry = false;
510         }
511         catch (InterruptedException e)
512         {
513             Log.e(TAG, "Thread interrupted", e);
514         }
515     }
516 } // fim do método surfaceDestroyed
517

```

Figura 6.18 Implementando os métodos de SurfaceHolder.Callback.

6.8.13 Sobrescrevendo o método onTouchEvent de View

Neste exemplo, sobrescrevemos o método onTouchEvent de View (Fig. 6.19) para determinar quando o usuário toca na tela. O parâmetro MotionEvent contém informações sobre o evento ocorrido. A linha 523 usa o método getAction de MotionEvent para determinar o tipo de evento de toque. Em seguida, as linhas 526 e 527 determinam se o usuário tocou na tela (MotionEvent.ACTION_DOWN) ou arrastou um dedo nela (MotionEvent.ACTION_MOVE). De qualquer modo, a linha 529 chama o método fireCannonball de cannonView para apontar e disparar o canhão na direção do ponto desse toque. Então, a linha 532 retorna true para indicar que o evento de toque foi tratado.

```

518 // chamado quando o usuário toca na tela nessa atividade
519 @Override
520 public boolean onTouchEvent(MotionEvent e)
521 {
522     // obtém valor int representando o tipo de ação que causou esse evento
523     int action = e.getAction();
524
525     // o usuário tocou na tela ou arrastou o dedo pela tela
526     if (action == MotionEvent.ACTION_DOWN ||
527         action == MotionEvent.ACTION_MOVE)
528     {
529         fireCannonball(e); // dispara a bala de canhão na direção do ponto do toque
530     }
531
532     return true;
533 } // fim do método onTouchEvent
534

```

Figura 6.19 Sobrescrevendo o método onTouchEvent de View.

6.8.14 CannonThread: usando uma thread para criar um loop de jogo

A Figura 6.20 define uma subclasse de Thread que atualiza o jogo. A thread mantém uma referência para SurfaceHolder de SurfaceView (linha 538) e um valor booleano que indica se a thread está *em execução*. O método run da classe (linhas 556 a 587) faz as *animações quadro a quadro* – isso é conhecido como *loop do jogo*. Cada atualização dos elementos do jogo na tela é feita com base no número de milissegundos decorridos desde a última atualização. A linha 559 obtém a hora atual do sistema, em milissegundos, quando a thread começa a ser executada. As linhas 561 a 586 fazem loop até que threadIsRunning seja false.

```

535 // subclasse de Thread para controlar o loop do jogo
536 private class CannonThread extends Thread
537 {
538     private SurfaceHolder surfaceHolder; // para manipular a tela de desenho
539     private boolean threadIsRunning = true; // executando por padrão
540
541     // inicializa holder de superfície
542     public CannonThread(SurfaceHolder holder)
543     {
544         surfaceHolder = holder;
545         setName("CannonThread");
546     }
547
548     // altera o estado de execução
549     public void setRunning(boolean running)
550     {
551         threadIsRunning = running;
552     }
553
554     // controla o loop do jogo
555     @Override
556     public void run()
557     {
558         Canvas canvas = null; // usado para desenhar
559         long previousFrameTime = System.currentTimeMillis();
560
561         while (threadIsRunning)
562         {
563             try
564             {
565                 // obtém objeto Canvas para desenho exclusivo a partir dessa thread
566                 canvas = surfaceHolder.lockCanvas(null);
567
568                 // bloqueia surfaceHolder para desenhar
569                 synchronized(surfaceHolder)
570                 {
571                     long currentTime = System.currentTimeMillis();
572                     double elapsedTimeMS = currentTime - previousFrameTime;
573                     totalElapsedTime += elapsedTimeMS / 1000.0;
574                     updatePositions(elapsedTimeMS); // atualiza o estado do jogo
575                     drawGameElements(canvas); // desenha usando a tela de desenho
576                     previousFrameTime = currentTime; // atualiza o tempo anterior
577                 }
578             }
579             finally
580             {

```

Figura 6.20 Objeto Runnable que atualiza o jogo a cada TIME_INTERVAL milissegundos.

```

581         // exibe o conteúdo da tela de desenho no componente CannonView
582         // e permite que outras threads utilizem o objeto Canvas
583         if (canvas != null)
584             surfaceHolder.unlockCanvasAndPost(canvas);
585     }
586 } // fim de while
587 } // fim do método run
588 } // fim da classe aninhada CannonThread
589 } // fim da classe CannonView

```

Figura 6.20 Objeto `Runnable` que atualiza o jogo a cada `TIME_INTERVAL` milissegundos.

Primeiramente, obtemos o objeto `Canvas` para desenhar no elemento `SurfaceView`, chamando o método `lockCanvas` de `SurfaceHolder` (linha 566). Somente uma thread por vez pode desenhar em um elemento `SurfaceView`. Para garantir isso, você deve primeiro *bloquear* o elemento `SurfaceHolder`, especificando-o como a expressão entre parênteses de um bloco `synchronized` (linha 569). Em seguida, obtemos o tempo atual, em milissegundos, e então calculamos o tempo decorrido e o somamos ao tempo total até o momento – isso vai ser usado para ajudar a exibir a quantidade de tempo restante no jogo. A linha 574 chama o método `updatePositions` para mover todos os elementos do jogo, passando como argumento o tempo decorrido, em milissegundos. Isso garante que o jogo opere na mesma velocidade, *independente do quanto o dispositivo seja rápido*. Se o tempo entre os quadros for maior (isto é, o dispositivo é mais lento), os elementos do jogo vão se mover mais quando cada quadro da animação for exibido. Se o tempo entre os quadros for menor (isto é, o dispositivo é mais rápido), os elementos do jogo vão se mover menos quando cada quadro da animação for exibido. Por fim, a linha 575 desenha os elementos do jogo usando o objeto `Canvas` de `SurfaceView`, e a linha 576 armazena o objeto `currentTime` como `previousFrameTime` a fim de se preparar para calcular o tempo decorrido entre esse quadro da animação e o *próximo*.

6.9 Para finalizar

Neste capítulo, você criou o aplicativo **Cannon Game**, o qual desafia o jogador a destruir um alvo de sete partes antes que um limite de tempo de 10 segundos expire. O usuário mira e dispara o canhão tocando na tela. Para desenhar na tela a partir de uma thread separada, você criou uma visualização personalizada, estendendo a classe `SurfaceView`. Você aprendeu que os nomes de classe de componentes personalizados devem ser totalmente qualificados no elemento de layout XML que representa o componente. Apresentamos mais métodos do ciclo de vida de `Fragment`. Você aprendeu que o método `onPause` é chamado quando um fragmento é pausado e que o método `onDestroy` é chamado quando o fragmento é destruído. Você tratou toques sobrescrevendo o método `onTouchEvent` de `View`. Adicionou efeitos sonoros à pasta `res/raw` do aplicativo e os gerenciou com um objeto `SoundPool`. Também usou o serviço `AudioManager` do sistema para obter o volume de música atual do dispositivo e o utilizou como volume de reprodução.

Este aplicativo realiza suas animações manualmente, atualizando os elementos do jogo em um componente `SurfaceView` de uma thread de execução separada. Para isso, você estendeu a classe `Thread` e criou um método `run` que exibe elementos gráficos chamando métodos da classe `Canvas`. Você usou o objeto `SurfaceHolder` da `SurfaceView` para obter o `Canvas` apropriado. Também aprendeu a construir um loop que controla um

jogo com base na quantidade de tempo decorrido entre quadros de animação, para que o jogo opere na mesma velocidade global em todos os dispositivos, independentemente da velocidade de seus processadores.

No Capítulo 7, apresentamos o aplicativo **Doodlz**, o qual utiliza recursos gráficos do Android para transformar a tela de um dispositivo em uma *tela de desenho virtual*. Você também vai aprender sobre o novo modo imersivo e sobre os recursos de impressão do Android 4.4.

Exercícios de revisão

- 6.1** Preencha os espaços em branco em cada um dos seguintes enunciados:
- a) Você pode criar uma visualização personalizada estendendo a classe `View` ou _____.
 - b) Para processar eventos de toque simples para uma atividade, você pode sobrescrever o método `onTouchEvent` da classe `Activity` e, então, usar constantes da classe _____ (pacote `android.view`) para testar qual tipo de evento ocorreu e processá-lo de forma correspondente.
 - c) Cada subclasse de `SurfaceView` deve implementar a interface _____, a qual contém métodos que são chamados quando o elemento `SurfaceView` é criado, alterado (por exemplo, em seu tamanho ou sua orientação) ou destruído.
 - d) O `d` em um especificador de formato indica que você está formatando um inteiro decimal, e o `f` em um especificador indica que você está formatando um valor _____.
 - e) Os arquivos de som são armazenados na pasta _____ do aplicativo.
- 6.2** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- a) A documentação do Android recomenda que os jogos utilizem o fluxo de áudio de música para reproduzir sons.
 - b) No Android, é importante maximizar o volume de trabalho feito na thread da interface gráfica para garantir que suas respostas permaneçam rápidas e que não sejam exibidas caixas de diálogo ANR (Application Not Responding).
 - c) Um Canvas desenha no objeto `Bitmap` de um componente `View`.
 - d) Os recursos `String` de formato que contêm vários especificadores de formato devem numerá-los para propósitos de adaptação ao idioma.
 - e) Existem sete fluxos de som, identificados pelas constantes na classe `AudioManager`, mas a documentação da classe `SoundPool` recomenda usar o fluxo para tocar música (`AudioManager.STREAM_MUSIC`) para som em jogos.
 - f) Os nomes de classe de componentes personalizados devem ser totalmente qualificados no elemento do layout XML que representa o componente.

Respostas dos exercícios de revisão

- 6.1** a) uma de suas subclasses. b) `MotionEvent`. c) `SurfaceHolder.Callback`. d) de ponto flutuante. e) `res/raw`.
- 6.2** a) Verdadeira. b) Falsa. No Android, é importante *minimizar* o volume de trabalho feito na thread da interface gráfica para garantir que suas respostas permaneçam rápidas e que não sejam exibidas caixas de diálogo ANR (Application Not Responding). c) Verdadeira. d) Verdadeira. e) Verdadeira. f) Verdadeira.

Exercícios

6.3 Preencha os espaços em branco em cada um dos seguintes enunciados:

- O método _____ é chamado para a atividade (Activity) atual quando outra atividade recebe o foco, a qual envia a atividade atual para o segundo plano.
- Quando uma atividade é encerrada, seu método _____ é chamado.
- Um _____ permite a um aplicativo reagir às interações mais sofisticadas do usuário, como movimentos rápidos, toques rápidos duplos, pressionamentos longos e rolagens.
- O método _____ de Activity especifica que o volume de um aplicativo pode ser controlado com as teclas de volume do dispositivo e deve ser igual ao volume de reprodução de música do dispositivo. O método recebe uma constante da classe AudioManager (pacote android.media).
- Os jogos frequentemente exigem lógica complexa que deve ser executada em threads de execução separadas, e essas threads muitas vezes precisam desenhar na tela. Para esses casos, o Android fornece a classe _____ – uma subclasse de View na qual qualquer thread pode desenhar.
- O método _____ é chamado para a atividade atual quando outra atividade recebe o foco.

6.4 Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.

- A classe SurfaceHolder também fornece métodos que dão a uma thread acesso compartilhado ao Canvas para desenhar, pois somente uma thread por vez pode desenhar em um elemento SurfaceView.
- MotionEvent.ACTION_TOUCH indica que o usuário tocou na tela e que moveu um dedo nela (MotionEvent.ACTION_MOVE).
- Quando um objeto View é inflado, seu construtor é chamado e recebe um objeto Context e um AttributeSet como argumentos.
- O método start de SoundPool recebe três argumentos – o objeto Context do aplicativo, um identificador de recurso que representa o arquivo de som a ser carregado e a prioridade do som.
- Quando um loop controla um jogo com base na quantidade de tempo decorrido entre quadros de animação, o jogo operará em velocidades diferentes, conforme for apropriado para cada dispositivo.

6.5 (*Aplicativo Cannon Game melhorado*) Modifique o aplicativo Cannon Game como segue:

- Use imagens para a base do canhão e para a bala.
- Exiba uma linha tracejada mostrando a trajetória da bala do canhão.
- Reproduza um som quando a barreira atingir a parte superior ou inferior da tela.
- Reproduza um som quando o alvo atingir a parte superior ou inferior da tela.
- Aprimore o aplicativo para que tenha nove níveis. Em cada nível, o alvo deve ter o mesmo número de partes.
- Mantenha um placar. Aumente a pontuação do usuário por 10 vezes o nível atual para cada parte do alvo atingida. Diminua a contagem por 15 vezes o nível atual sempre que o usuário atingir a barreira. Exiba a pontuação mais alta no canto superior esquerdo da tela.
- Salve as cinco pontuações mais altas em um arquivo SharedPreferences. Quando o jogo terminar, exiba um componente AlertDialog com as pontuações mostradas em ordem decrescente. Se a pontuação do usuário for uma das cinco mais altas, destaque essa pontuação exibindo um asterico (*) ao lado dela.

- h) Adicione uma animação de explosão sempre que a bala do canhão atingir uma das partes do alvo.
- i) Adicione uma animação de explosão sempre que a bala do canhão atingir a barreira.
- j) Quando a bala do canhão atingir a barreira, aumente o comprimento dela em 5%.
- k) Torne o jogo mais difícil à medida que avançar, aumentando a velocidade do alvo e da barreira.
- l) Adicione funcionalidade para vários jogadores, permitindo que dois deles joguem no mesmo dispositivo.
- m) Aumente o número de obstáculos entre o canhão e o alvo.
- n) Adicione uma rodada de bônus com duração de quatro segundos. Mude a cor do alvo e acrescente música para indicar que se trata de uma rodada de bônus. Se o usuário atingir uma parte do alvo durante esses quatro segundos, dê a ele um bônus de 1000 pontos.

6.6 (*Aplicativo Brick Game*) Crie um jogo semelhante ao do canhão, mas que atire projéteis em uma parede de tijolos fixa. O objetivo é destruir o suficiente da parede para atirar no alvo móvel atrás dela. Quanto mais rápido você abrir uma brecha na parede e alcançar o alvo, maior sua pontuação. Varie a cor dos tijolos e o número de tiros exigidos para destruir cada um — por exemplo, tijolos vermelhos podem ser destruídos com três tiros, amarelos podem ser destruídos com seis tiros, etc. Inclua várias camadas na parede e um alvo móvel pequeno (por exemplo, um ícone, um animal, etc.). Mantenha um placar. Aumente a dificuldade a cada rodada, adicionando mais camadas à parede e aumentando a velocidade do alvo móvel.

6.7 (*Aplicativo para Tablet: Horse Race para vários jogadores com Cannon Game*) Um dos jogos de azar mais populares é o de corrida de cavalos. Cada jogador escolhe um cavalo. Para mover o cavalo, os jogadores devem mostrar uma habilidade — como atirar em um alvo através de uma corrente de água. Sempre que um jogador atinge um alvo, seu cavalo avança. O objetivo é atingir o alvo o máximo de vezes e o mais rápido possível, para mover o cavalo até a linha de chegada e vencer a corrida.

Crie um aplicativo para tablet, para vários jogadores, que simule o jogo *Horse Race* com dois jogadores. Em vez de uma corrente de água, use o jogo *Cannon Game* como a habilidade que move cada cavalo. Sempre que um jogador atinge uma parte do alvo com o canhão, move seu cavalo uma posição para a direita.

Configure a orientação da tela como paisagem e use a API nível 11 (Android 3.0) ou superior, para que o jogo funcione em tablets. Divida a tela em três seções. A primeira deve abranger toda a largura da parte superior da tela — essa será a pista de corrida. Abaixo da pista, inclua duas seções lado a lado. Em cada uma delas, inclua jogos *Cannon Game* separados. Os dois jogadores precisarão estar lado a lado para jogar esta versão do jogo.

Na pista de corrida, inclua dois cavalos que começam na esquerda e se movem para a direita em direção a uma linha de chegada no lado direito da tela. Numere os cavalos como “1” e “2”.

Inclua muitos sons de uma corrida de cavalos tradicional. Você pode encontrar áudios gratuitos online em sites como www.audiomicro.com/ ou criar os seus próprios. Antes da corrida, reproduza um áudio do toque de corneta tradicional — o “Call to Post” — significando que os cavalos devem ficar em suas marcas. Inclua o som do tiro de partida da corrida, seguido do locutor dizendo “Foi dada a largada!”.

6.8 (*Aplicativo Bouncing Ball Game*) Crie um aplicativo de jogo no qual o objetivo do usuário é evitar que uma bola saltitante caia na parte inferior da tela. Quando o usuário pressiona o botão de início, uma bola quica na parte superior esquerda e direita (as “paredes”) da tela. Uma barra horizontal embaixo na tela serve como raquete para impedir que a bola atinja

a parte inferior da tela. (A bola pode quicar na raquete, mas não na parte inferior da tela.) Permita que o usuário arraste a raquete para a esquerda e para a direita. Se a bola colide com a raquete, ela quica para cima e o jogo continua. Se a bola atinge a parte inferior, o jogo termina. Diminua a largura da raquete a cada 20 segundos e aumente a velocidade da bola para tornar o jogo mais desafiador. Pense na possibilidade de acrescentar obstáculos em lugares aleatórios.

- 6.9** (*Aplicativo Digital Clock*) Crie um aplicativo que mostre um relógio digital na tela. Inclua funcionalidade de despertador.
- 6.10** (*Aplicativo Analog Clock*) Crie um aplicativo que mostre um relógio analógico com ponteiros de hora, minuto e segundo que se movam apropriadamente à medida que o tempo passar.
- 6.11** (*Aplicativo Fireworks Designer*) Crie um aplicativo que permita ao usuário criar uma tela com fogos de artifício personalizada. Crie uma variedade de demonstrações de fogos de artifício. Então, organize o disparo dos fogos de artifício para obter o efeito máximo. Você pode sincronizar seus fogos de artifício com áudios ou vídeos. Pode também sobrepor-los em uma imagem.
- 6.12** (*Aplicativo Towers of Hanoi animada*) Todo cientista da computação iniciante deve atacar certos problemas clássicos, e o Towers of Hanoi (Torres de Hanói) (consulte a Fig. 6.21) é um dos mais famosos. A lenda diz que, em um templo no Extremo Oriente, os sacerdotes estão tentando mover uma pilha de discos de um pino para outro. A pilha inicial tem 64 discos enfiados em um pino e organizados de baixo para cima por tamanho decrescente. Os sacerdotes estão tentando mover a pilha desse pino para um segundo pino, com as restrições de que exatamente um disco é movido por vez e em nenhum momento um disco maior pode ser colocado em cima de um disco menor.

Existe um terceiro pino para conter discos temporariamente. Supostamente, o mundo acabará quando os sacerdotes completarem sua tarefa; portanto, somos pouco incentivados a facilitar seu trabalho.

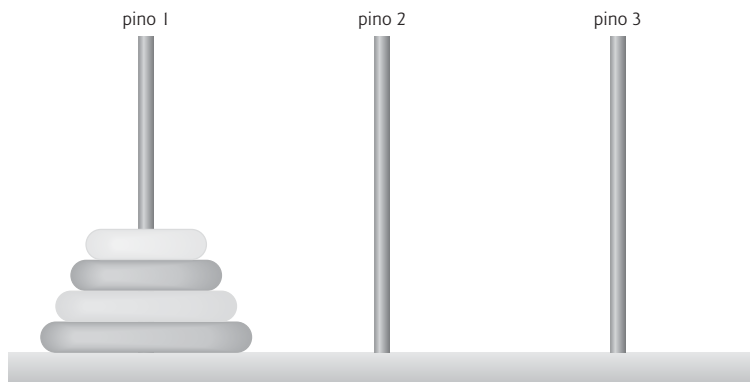


Figura 6.21 As Towers of Hanoi (Torres de Hanói) para o caso com quatro discos.

Vamos supor que os sacerdotes estejam tentando mover os discos do pino 1 para o pino 3. Queremos desenvolver um algoritmo que exiba a sequência precisa das transferências dos discos de um pino para outro.

Se fôssemos encarar esse problema com métodos convencionais, rapidamente perderíamos a esperança, confusos no gerenciamento dos discos. Em vez disso, se atacarmos o

problema pensando na recursividade, ele se torna tratável imediatamente. A movimentação de n discos pode ser vista em termos de mover apenas $n - 1$ discos (daí a recursividade), como segue:

- a) Mova $n - 1$ discos do pino 1 para o pino 2, usando o pino 3 como área de armazenamento temporário.
- b) Mova o último disco (o maior) do pino 1 para o pino 3.
- c) Mova $n - 1$ discos do pino 2 para o pino 3, usando o pino 1 como área de armazenamento temporário.

O processo termina quando a última tarefa envolve mover $n = 1$ disco (isto é, o caso básico). Essa tarefa é realizada simplesmente movendo-se o disco, sem a necessidade de uma área de armazenamento temporário.

Escreva um aplicativo para resolver o problema das Towers of Hanoi (Torres de Hanói). Permita que o usuário digite o número de discos. Use um método Tower recursivo com quatro parâmetros:

- a) o número de discos a serem movidos
- b) o pino no qual esses discos são enfiados inicialmente
- c) o pino para o qual essa pilha de discos deve ser movida
- d) o pino a ser usado como área de armazenamento temporário

Seu aplicativo deve exibir as instruções precisas necessárias para mover os discos do pino inicial para o de destino, e deve mostrar animações dos discos se movendo de um pino para outro. Por exemplo, para mover uma pilha de três discos do pino 1 para o pino 3, seu aplicativo deve exibir a seguinte série de movimentos e as animações correspondentes:

```
1 --> 3 (Esta notação significa "Mova um disco do pino 1 para o pino 3".)
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
```



Objetivos

Neste capítulo, você vai:

- Detectar quando o usuário toca na tela, move o dedo pela tela e retira o dedo da tela.
- Processar múltiplos toques na tela para que o usuário possa desenhar simultaneamente com vários dedos.
- Usar um componente `SensorManager` e o acelerômetro para detectar eventos de movimento.
- Usar um objeto `AtomicBoolean` para que múltiplos threads acessem um valor booleano de forma segura para threads.
- Usar um objeto `Paint` para especificar a cor e a largura de uma linha.
- Usar objetos `Path` para armazenar os dados de cada linha e um objeto `Canvas` para desenhar cada linha em um componente `Bitmap`.
- Criar um menu e exibir itens de menu na barra de ação.
- Usar o modo imersivo do Android 4.4 para permitir ao usuário desenhar na tela inteira.
- Usar o framework de impressão do Android 4.4 e a classe `PrintHelper` da Android Support Library para permitir ao usuário imprimir um desenho.

- 7.1 Introdução
- 7.2 Visão geral das tecnologias
 - 7.2.1 Uso de `SensorManager` para detectar eventos de acelerômetro
 - 7.2.2 Componentes `DialogFragment` personalizados
 - 7.2.3 Desenho com `Canvas` e `Bitmap`
 - 7.2.4 Processamento de múltiplos eventos de toque e armazenamento de linhas em objetos `Path`
 - 7.2.5 Modo imersivo do Android 4.4
 - 7.2.6 `GestureDetector` e `SimpleOnGestureListener`
 - 7.2.7 Salvando o desenho na galeria do dispositivo
 - 7.2.8 Impressão no Android 4.4 e a classe `PrintHelper` da Android Support Library
- 7.3 Construção da interface gráfica do usuário e arquivos de recurso do aplicativo
 - 7.3.1 Criação do projeto
 - 7.3.2 `strings.xml`
 - 7.3.3 `dimens.xml`
 - 7.3.4 Menu do componente `DoodleFragment`
 - 7.3.5 Layout de `activity_main.xml` para `MainActivity`
 - 7.3.6 Layout de `fragment_doodle.xml` para `DoodleFragment`
 - 7.3.7 Layout de `fragment_color.xml` para `ColorDialogFragment`
 - 7.3.8 Layout de `fragment_line_width.xml` para `LineWidthDialogFragment`
 - 7.3.9 Adição da classe `EraseImageDialogFragment`
- 7.4 Classe `MainActivity`
- 7.5 Classe `DoodleFragment`
- 7.6 Classe `DoodleView`
- 7.7 Classe `ColorDialogFragment`
- 7.8 Classe `LineWidthDialogFragment`
- 7.9 Classe `EraseImageDialogFragment`
- 7.10 Para finalizar

Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios

7.1 Introdução

O aplicativo **Doodlz** (Fig. 7.1) permite que você pinte arrastando um ou mais dedos pela tela. Ele utiliza o *modo imersivo* do Android 4.4 para que você possa desenhar na tela inteira – as *barras de sistema* e a *barra de ação* do dispositivo alternam entre estarem visíveis e ocultas quando você dá um toque rápido na tela.

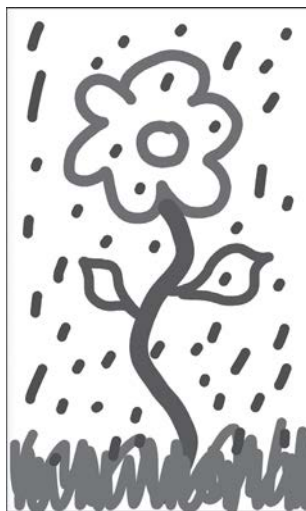
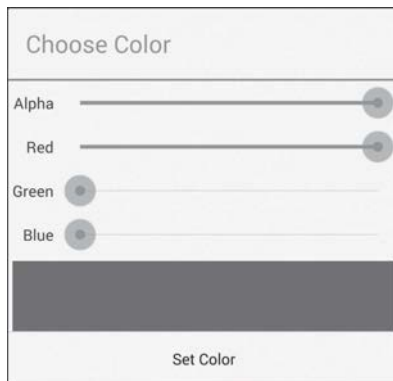
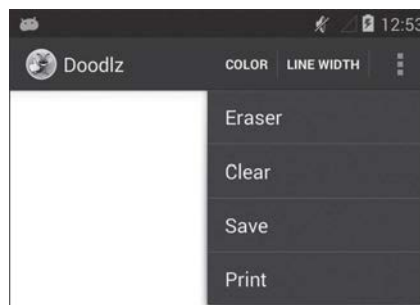


Figura 7.1 Aplicativo **Doodlz** com um desenho finalizado.

As opções do aplicativo permitem configurar a cor do desenho e a largura da linha. A caixa de diálogo **Choose Color** (Fig. 7.2(a)) fornece componentes SeekBar (isto é, controles deslizantes) para *alfa* (*transparência*), vermelho, verde e azul que permitem selecionar a cor ARGB (apresentada na Seção 1.9). Quando você move o cursor de cada componente SeekBar, a cor atualizada aparece abaixo dele. A caixa de diálogo **Choose Line Width** (Fig. 7.2(b)) fornece um componente SeekBar que controla a espessura da linha que vai ser desenhada. Outros itens no *menu de opções* do aplicativo (Fig. 7.3) permitem transformar seu dedo em uma borracha (**Eraser**), limpar a tela (**Clear**), salvar o desenho atual (**Save**) na galeria (**Gallery**) de seu dispositivo e, em dispositivos Android 4.4, imprimir o desenho atual. Dependendo do tamanho da tela de seu dispositivo, alguns ou todos os itens de menu do aplicativo aparecem diretamente na barra de ação – os que não couberem serão exibidos no menu de opções. A qualquer momento, você pode *chacoalhar* o dispositivo para apagar todo o desenho da tela. Você testou este aplicativo na Seção 1.9, de modo que não apresentaremos um teste neste capítulo. Embora ele funcione em AVDs, os recursos são mais fluidos em dispositivos reais. [Obs.: Devido a um erro no aplicativo **Gallery** quando este livro estava sendo produzido, em alguns dispositivos talvez seja necessário tirar uma foto com a câmera do aparelho antes que você possa salvar corretamente a partir do aplicativo **Doodlz**.]

a) Caixa de diálogo **Choose Color**b) Caixa de diálogo **Choose Line Width****Figura 7.2** Caixas de diálogo **Choose Color** e **Choose Line Width** do aplicativo **Doodlz**.**Figura 7.3** Opções de menu adicionais do aplicativo **Doodlz**, conforme aparecem em um telefone Android 4.4.

7.2 Visão geral das tecnologias

Esta seção apresenta as novas tecnologias que usamos no aplicativo Doodlz.

7.2.1 Uso de `SensorManager` para detectar eventos de acelerômetro

Neste aplicativo, você pode chacoalhar o dispositivo para apagar o desenho atual. A maioria dos dispositivos tem um **acelerômetro** que permite aos aplicativos detectar movimentos. Outros sensores atualmente suportados pelo Android incluem gravidade, giroscópio, luz, aceleração linear, campo magnético, orientação, pressão, proximidade, vetor de rotação e temperatura. A lista de constantes de **Sensor** que representam esses tipos de sensor pode ser encontrada no endereço:

<http://developer.android.com/reference/android/hardware/Sensor.html>

Na Seção 7.5, vamos discutir o tratamento de eventos de acelerômetro e sensor. Para ver uma discussão completa sobre outros sensores do Android, consulte o *Sensors Overview*, no endereço

http://developer.android.com/guide/topics/sensors/sensors_overview.html

7.2.2 Componentes `DialogFragment` personalizados

Vários aplicativos têm usado elementos `AlertDialog` em componentes `DialogFragment` para exibir informações para o usuário ou para fazer perguntas e receber respostas na forma de cliques em componentes `Button`. Os elementos `AlertDialog` que você usou até aqui foram criados com classes internas anônimas que estendiam `DialogFragment` e exibiam apenas texto e botões. Os elementos `AlertDialog` também podem conter objetos `View` personalizados. Neste aplicativo, você vai definir três subclasses de `DialogFragment`:

- `ColorDialogFragment` (Seção 7.7) exibe um elemento `AlertDialog` com um objeto `View` personalizado que contém componentes de interface gráfica do usuário para visualizar e selecionar uma nova cor de desenho `ARGB`.
- `LineWidthDialogFragment` (Seção 7.8) exibe um elemento `AlertDialog` com um objeto `View` personalizado que contém componentes de interface gráfica do usuário para visualizar e selecionar a espessura da linha.
- `EraseImageDialogFragment` (Seção 7.9) exibe um elemento `AlertDialog` padrão solicitando ao usuário para que confirme se a imagem inteira deve ser apagada.

Para `ColorDialogFragment` e `EraseImageDialogFragment`, você vai inflar o objeto `View` personalizado a partir de um arquivo de recurso de layout. Em cada uma das três subclasses de `DialogFragment`, vai também sobrescrever os seguintes métodos de ciclo de vida de `Fragment`:

- **onAttach** – O primeiro método de ciclo de vida de `Fragment` chamado quando um fragmento é anexado a uma atividade pai.
- **onDetach** – O último método de ciclo de vida de `Fragment` chamado quando um fragmento está para ser desanexado de uma atividade pai.

Impedindo que várias caixas de diálogo apareçam ao mesmo tempo

É possível que a rotina de tratamento do evento de chacoalhar tente exibir a caixa de diálogo de confirmação para apagar uma imagem quando outra caixa de diálogo já estiver na tela. Para evitar isso, você vai usar `onAttach` e `onDetach` para configurar o valor de

uma variável booleana que indica se uma caixa de diálogo está na tela. Quando o valor da variável `boolean` é `true`, não permitimos que a rotina de tratamento do evento de chacoalhar exiba uma caixa de diálogo.

7.2.3 Desenho com Canvas e Bitmap

Este aplicativo desenha linhas em objetos **Bitmap** (pacote `android.graphics`). Você pode associar um **Canvas** a um **Bitmap** e, então, usar o **Canvas** para fazer um desenho no **Bitmap**, o qual pode então ser exibido na tela (Seção 7.6). Um objeto **Bitmap** também pode ser salvo em um arquivo – vamos usar essa capacidade para armazenar desenhos na galeria do dispositivo quando o usuário tocar na opção **Save**.

7.2.4 Processamento de múltiplos eventos de toque e armazenamento de linhas em objetos Path

Você pode arrastar um ou mais dedos pela tela para desenhar. O aplicativo armazena a informação de *cada* dedo como um objeto **Path** (pacote `android.graphics`) que representa segmentos de linha e curvas. Os *eventos de toque* são processados sobrescrevendo-se o método `onTouchEvent` de **View** (Seção 7.6). Esse método recebe um objeto **MotionEvent** (pacote `android.view`) que contém o tipo de evento de toque ocorrido e o identificador do dedo (isto é, ponteiro) que gerou o evento. Usamos identificadores (IDs) para distinguir os diferentes dedos e adicionar informações aos objetos **Path** correspondentes. Usamos o tipo do evento de toque para determinar se o usuário *tocou* na tela, *arrastou* pela tela ou *tirou um dedo* da tela.

7.2.5 Modo imersivo do Android 4.4

O Android 4.4 apresenta um novo **modo imersivo** (Seção 7.6), o qual permite a um aplicativo utilizar a tela inteira, mas ainda permite que o usuário possa acessar as barras de sistema quando necessário. Você vai usar esse modo quando este aplicativo estiver sendo executado em um dispositivo Android 4.4 ou mais recente.

7.2.6 GestureDetector e SimpleOnGestureListener

Este aplicativo utiliza um componente **GestureDetector** (pacote `android.view`) para ocultar ou exibir as barras de sistema do dispositivo e sua barra de ação. Um componente **GestureDetector** permite a um aplicativo reagir às interações do usuário, como *movimentos rápidos* (*swipes*), *toques rápidos*, *toques rápidos duplos*, *pressionamentos longos* e *rolagens*, implementando os métodos das interfaces **GestureDetector.OnGestureListener** e **GestureDetector.OnDoubleTapListener**. A classe **GestureDetector.SimpleOnGestureListener** é uma *classe adaptadora* que implementa todos os métodos dessas duas interfaces, de modo que você pode estender essa classe e sobrescrever apenas o método (ou métodos) necessário dessas interfaces. Na Seção 7.6, você vai inicializar um componente **GestureDetector** com um objeto **SimpleOnGestureListener**, o qual vai tratar o evento de *toque rápido* que oculta ou exibe as barras de sistema e a barra de ação.

7.2.7 Salvando o desenho na galeria do dispositivo

O aplicativo fornece uma opção **Save** que permite ao usuário salvar um desenho na galeria do dispositivo – o local padrão no qual são armazenadas as fotos tiradas com o dispositivo. Um objeto **ContentResolver** (pacote `android.content`) permite ao aplicativo

ler e armazenar dados em um dispositivo. Você vai usar um objeto `ContentResolver` (Seção 7.6) e o método `insertImage` da classe `MediaStore.Images.Media` para salvar uma imagem na galeria do dispositivo. O componente **MediaStore** gerencia os arquivos de mídia (imagens, áudio e vídeo) armazenados em um dispositivo.

7.2.8 Impressão no Android 4.4 e a classe `PrintHelper` da **Android Support Library**

Agora o Android 4.4 inclui um framework de impressão. Neste aplicativo, usamos a classe `PrintHelper` (Seção 7.6) para imprimir o desenho atual. A classe `PrintHelper` fornece uma interface de usuário para selecionar uma impressora, tem um método para determinar se um dispositivo suporta impressão e fornece um método para imprimir um objeto `Bitmap`. `PrintHelper` faz parte da *Android Support Library* – um conjunto de bibliotecas comumente usadas a fim de fornecer novos recursos Android para uso em versões mais antigas do Android. As bibliotecas também contêm recursos de conveniência adicionais, como a classe `PrintHelper`, que suporta versões específicas do Android.

7.3 Construção da interface gráfica do usuário e arquivos de recurso do aplicativo

Nesta seção, você vai criar os arquivos de recurso, os arquivos de layout da interface gráfica do usuário e as classes do aplicativo **Doodlz**.

7.3.1 Criação do projeto

Comece criando um novo projeto Android chamado **Doodlz**. Especifique os valores a seguir na caixa de diálogo **New Android Project** e, em seguida, pressione **Finish**:

- **Application Name:** Doodlz
- **Project Name:** Doodlz
- **Package Name:** com.deitel.doodlz
- **Minimum Required SDK:** API18: Android 4.3
- **Target SDK:** API19: Android 4.4
- **Compile With:** API19: Android 4.4
- **Theme:** **Holo Light with Dark Action Bar**

No segundo passo de **New Android Application** da caixa de diálogo **New Android Project**, deixe as configurações padrão e pressione **Next >**. No passo **Configure Launcher Icon**, selecione uma imagem de ícone de aplicativo e, então, pressione **Next >**. No passo **Create Activity**, selecione **Blank Activity** e pressione **Next >**. No passo **Blank Activity**, deixe as configurações padrão e clique em **Finish** para criar o projeto. Abra `activity_main.xml` no editor **Graphical Layout** e selecione **Nexus 4** na lista suspensa de tipo de tela. Mais uma vez, usaremos esse dispositivo como base para nosso projeto.

O novo projeto será automaticamente configurado para usar a versão atual da **Android Support Library**. Caso esteja atualizando um projeto já existente, você pode adicionar a ele a versão mais recente da **Android Support Library**. Para ver os detalhes, visite:

<http://developer.android.com/tools/support-library/index.html>
<http://developer.android.com/tools/support-library/setup.html>

7.3.2 strings.xml

Você criou recursos de String em capítulos anteriores, de modo que mostramos aqui apenas uma tabela dos nomes de recursos de String e valores correspondentes (Fig. 7.4). Clique duas vezes em `strings.xml` na pasta `res/values` a fim de exibir o editor de recursos para criar esses recursos de String.

Nome do recurso	Valor
<code>app_name</code>	Doodlz
<code>button_erase</code>	Erase Image
<code>button_cancel</code>	Cancel
<code>button_set_color</code>	Set Color
<code>button_set_line_width</code>	Set Line Width
<code>line_imageview_description</code>	This displays the line thickness
<code>label_alpha</code>	Alpha
<code>label_red</code>	Red
<code>label_green</code>	Green
<code>label_blue</code>	Blue
<code>menuitem_clear</code>	Clear
<code>menuitem_color</code>	Color
<code>menuitem_eraser</code>	Eraser
<code>menuitem_line_width</code>	Line Width
<code>menuitem_save</code>	Save
<code>menuitem_print</code>	Print
<code>message_erase</code>	Erase the drawing?
<code>message_error_saving</code>	There was an error saving the image
<code>message_saved</code>	Your painting has been saved to the Gallery
<code>message_error_printing</code>	Your device does not support printing
<code>title_color_dialog</code>	Choose Color
<code>title_line_width_dialog</code>	Choose Line Width

Figura 7.4 Recursos de String usados no aplicativo Doodlz.

7.3.3 dims.xml

A Figura 7.5 mostra uma tabela dos nomes e valores de recurso de dimensão que adicionamos a `dims.xml`. Abra `dims.xml` na pasta `res/values` a fim de exibir o editor de recursos para criar esses recursos. O recurso `line_imageview_height` especifica a altura do objeto `ImageView` que mostra a largura da linha no componente `LineWidthDialogFragment`, e o recurso `color_view_height` especifica a altura do objeto `View` que mostra a cor de desenho no componente `ColorDialogFragment`.

Nome do recurso	Valor
<code>line_imageview_height</code>	50dp
<code>color_view_height</code>	80dp

Figura 7.5 Recursos de dimensão usados no aplicativo Doodlz.

7.3.4 Menu do componente DoodleFragment

No Capítulo 5, você usou o menu padrão fornecido pelo IDE para exibir o item de menu **Settings** do aplicativo **Flag Quiz**. No Doodlez, não vamos usar o menu padrão, de modo que você pode excluir o arquivo `main.xml` na pasta `res/menu` de seu projeto; você vai definir seu próprio menu para o componente `DoodleFragment`.

Menus para diferentes versões de Android

Você vai fornecer duas versões do menu de `DoodleFragment` – uma para dispositivos Android 4.3 e anteriores e outra para dispositivos Android 4.4 e posteriores. A impressão só está disponível no Android 4.4 e posteriores; portanto, apenas o menu para esses dispositivos vai conter uma opção **Print**. Para permitir menus distintos, você vai definir um recurso de menu na pasta `res/menu` e outro separado na pasta `res/menu-v19` – 19 é a versão da API do Android correspondente ao Android 4.4. O Android vai escolher o recurso de menu na pasta `res/menu-v19` quando o aplicativo estiver sendo executado em dispositivos Android 4.4 e posteriores. Para criar a pasta `res/menu-v19`, clique com o botão direito do mouse na pasta `res`, selecione **New > Folder**, especifique `menu-v19` para **Folder name** e clique em **Finish**.

Menu para Android 4.3 e versões anteriores

Para criar o recurso de menu para Android 4.3 e versões anteriores:

1. Clique com o botão direito do mouse na pasta `res/menu` e selecione **New > Android XML File**.
2. Na caixa de diálogo que aparece, chame o arquivo de `doodle_fragment_menu.xml` e clique em **Finish**. O IDE abre o arquivo no editor de recursos de menu.
3. Clique em **Add...**, clique na guia **Layout** na caixa de diálogo do editor que aparece, selecione **Item** e clique em **OK**. O IDE realça o novo item e exibe seus atributos à direita.
4. Altere suas propriedades **Id** para `@+id/color`, **Title** para `@string/menuitem_color` e **Show as action** para `ifRoom`. O valor `ifRoom` indica que o Android deve exibir o item de menu na barra de ação, caso haja espaço; caso contrário, o item aparecerá no menu de opções, à direita da barra de ação. Outros valores de **Show as action** podem ser encontrados em <http://developer.android.com/guide/topics/resources/menu-resource.html>.
5. Repita os passos 3 e 4 para os itens `linewidht`, `eraser`, `clear` e `save` da Fig. 7.6. Observe que, quando você clicar em **Add...** para cada item de menu adicional, precisará selecionar **Create a new element at the top level in Menu** na caixa de diálogo que aparece.
6. Salve e feche `doodle_fragment_menu.xml`.

Id	Título
<code>@+id/linewidht</code>	<code>@string/menuitem_line_width</code>
<code>@+id/eraser</code>	<code>@string/menuitem_eraser</code>
<code>@+id/clear</code>	<code>@string/menuitem_clear</code>
<code>@+id/save</code>	<code>@string/menuitem_save</code>

Figura 7.6 Itens de menu adicionais para o componente `DoodleFragment`.

Menu para Android 4.4 e versões posteriores

Para criar o recurso de menu para dispositivos Android 4.4 e posteriores:

1. Copie `doodle_fragment_menu.xml` de `res/menu`, cole em `res/menu-v19` e abra o arquivo.
2. Clique em **Add...**, selecione **Create a new element at the top level in Menu** na caixa de diálogo que aparece, selecione **Item** e clique em **OK**.
3. No novo item, altere as propriedades **Id** para `@+id/print`, **Title** para `@string/menuitem_print` e **Show as action** para `ifRoom`.

7.3.5 Layout de `activity_main.xml` para `MainActivity`

O layout de `activity_main.xml` para o componente `MainActivity` deste aplicativo contém apenas o elemento `DoodleFragment`. Para adicionar esse fragmento ao layout:

1. Abra `activity_main.xml` no editor **Graphical Layout** e siga os passos da Seção 2.5.2 para mudar de `FrameLayout` para `RelativeLayout`.
2. Da seção **Layouts** da **Palette**, arraste um objeto **Fragment** para a área de projeto ou para o nó `RelativeLayout` na janela **Outline**.
3. O passo anterior exibe a caixa de diálogo **Choose Fragment Class**. Clique em **Create New...** para exibir a caixa de diálogo **New Java Class**.
4. Digite `DoodleFragment` no campo **Name** da caixa de diálogo, mude o valor do campo **Superclass** para `android.app.Fragment` e clique em **Finish** para criar a classe. O IDE abre o arquivo Java da classe, o qual você pode fechar por enquanto.
5. Altere a propriedade **Id** do novo fragmento para `@+id/doodleFragment` e salve o layout.

7.3.6 Layout de `fragment_doodle.xml` para `DoodleFragment`

O layout de `fragment_doodle.xml` para o componente `DoodleFragment` contém um objeto `FrameLayout` que exibe o elemento `DoodleView`. Nesta seção, você vai criar o layout de `DoodleFragment` e a classe `DoodleView`. Para adicionar o layout de `fragment_doodle.xml`:

1. Expanda o nó `res/layout` do projeto no **Package Explorer**.
2. Clique com o botão direito do mouse na pasta `layout` e selecione **New > Android XML File** para exibir a caixa de diálogo **New Android XML File**.
3. No campo **File** da caixa de diálogo, digite `fragment_doodle.xml`.
4. Na seção **Root Element**, selecione `FrameLayout` e, então, clique em **Finish**.
5. Da seção **Advanced** da **Palette**, arraste um objeto **view** (com **v** minúsculo) para a interface gráfica do usuário.
6. O passo anterior exibe a caixa de diálogo **Choose Custom View Class**. Nessa caixa de diálogo, clique em **Create New...** para exibir a caixa de diálogo **New Java Class**.
7. No campo **Name**, digite `DoodleView`. Certifique-se de que **Constructor from superclass** esteja selecionado e, em seguida, clique em **Finish**. Isso cria e abre `DoodleView.java`. Vamos usar somente o construtor de dois argumentos; portanto, exclua os outros dois. Salve e feche `DoodleView.java`.
8. Em `fragment_doodle.xml`, selecione `view1` na janela **Outline**. Na seção **Layout Parameters** da janela **Properties**, configure **Width** e **Height** como `match_parent`.

9. Na janela **Outline**, clique com o botão direito do mouse em **view1**, selecione **Edit ID...**, mude o nome de **view1** para **doodleView** e clique em **OK**.
10. Salve e feche **fragment_doodle.xml**.

7.3.7 Layout de **fragment_color.xml** para **ColorDialogFragment**

O layout de **fragment_color.xml** para o componente **ColorDialogFragment** contém um elemento **GridLayout** que exibe uma interface gráfica do usuário para selecionar e visualizar uma nova cor de desenho. Nesta seção, você vai criar o layout de **ColorDialogFragment** e a classe **ColorDialogFragment**. Para adicionar o layout de **fragment_color.xml**:

1. Expanda o nó **res/layout** do projeto no **Package Explorer**.
2. Clique com o botão direito do mouse na pasta **layout** e selecione **New > Android XML File** para exibir a caixa de diálogo **New Android XML File**.
3. No campo **File** da caixa de diálogo, digite **fragment_color.xml**.
4. Na seção **Root Element**, selecione **GridLayout** e, então, clique em **Finish**.
5. Na janela **Outline**, selecione o componente **GridLayout** e altere o valor de sua propriedade **id** para **@+id/colorDialogGridLayout**.
6. Usando a **Palette** do editor **Graphical Layout**, arraste componentes **TextView**, **SeekBar** e um objeto **View** para o nó **colorDialogGridLayout** na janela **Outline**. Arraste os itens na ordem em que estão listados na Fig. 7.7 e configure a propriedade **id** de cada item como mostrado na figura.

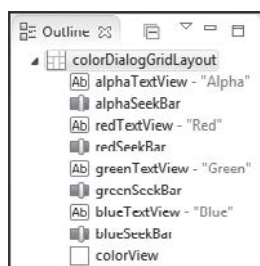


Figura 7.7 Visualização **Outline** para **fragment_color.xml**.

7. Depois de terminar o passo 6, configure as propriedades dos componentes da interface gráfica do usuário com os valores mostrados na Fig. 7.8 e, então, salve e feche **fragment_color.xml**.

Componente da interface gráfica do usuário	Propriedade	Valor
colorDialogGridLayout	Column Count	2
	Orientation	vertical
	Use Default Margins	true

Figura 7.8 Valores de propriedade para os componentes da interface gráfica do usuário em **fragment_color.xml**. (continua)

Componente da interface gráfica do usuário	Propriedade	Valor
alphaTextView	<i>Parâmetros do layout</i>	
	Column	0
	Gravity	right center_vertical
	Row	0
	<i>Outras propriedades</i>	
alphaSeekBar	Text	@string/label_alpha
	<i>Parâmetros do layout</i>	
	Column	1
	Gravity	fill_horizontal
	Row	0
redTextView	<i>Outras propriedades</i>	
	Max	255
	<i>Parâmetros do layout</i>	
	Column	0
	Gravity	right center_vertical
redSeekBar	Row	1
	<i>Outras propriedades</i>	
	Text	@string/label_red
	<i>Parâmetros do layout</i>	
	Column	1
greenTextView	Gravity	fill_horizontal
	Row	1
	<i>Outras propriedades</i>	
	Max	255
	<i>Parâmetros do layout</i>	
greenSeekBar	Column	0
	Gravity	right center_vertical
	Row	2
	<i>Outras propriedades</i>	
	Text	@string/label_green
blueTextView	<i>Parâmetros do layout</i>	
	Column	1
	Gravity	fill_horizontal
	Row	2
	<i>Outras propriedades</i>	
blueSeekBar	Max	255
	<i>Parâmetros do layout</i>	
	Column	0
	Gravity	right center_vertical
	Row	3
blueSeekBar	<i>Outras propriedades</i>	
	Text	@string/label_blue

Figura 7.8 Valores de propriedade para os componentes da interface gráfica do usuário em `fragment_color.xml`. (continua)

Componente da interface gráfica do usuário	Propriedade	Valor
blueSeekBar	<i>Parâmetros do layout</i>	
	Column	1
	Gravity	fill_horizontal
	Row	3
	<i>Outras propriedades</i>	
	Max	255
colorView	<i>Parâmetros do layout</i>	
	Height	@dimen/color_view_height
	Column	0
	Column Span	2
	Gravity	fill_horizontal

Figura 7.8 Valores de propriedade para os componentes da interface gráfica do usuário em `fragment_color.xml`.

Adicionando a classe `ColorDialogFragment` ao projeto

Para adicionar a classe `ColorDialogFragment` ao projeto:

1. Clique com o botão direito do mouse no pacote `com.deitel.doodlz` na pasta `src` do projeto e selecione **New > Class** para exibir a caixa de diálogo **New Java Class**.
2. No campo **Name**, digite `ColorDialogFragment`.
3. No campo **Superclass**, mude a superclasse para `android.app.DialogFragment`.
4. Clique em **Finish** para criar a classe.

7.3.8 Layout de `fragment_line_width.xml` para `LineWidthDialogFragment`

O layout de `fragment_line_width.xml` para o componente `LineWidthDialogFragment` contém um elemento `GridLayout` que exibe uma interface gráfica do usuário para selecionar e visualizar uma nova espessura de linha. Nesta seção, você vai criar o layout de `LineWidthDialogFragment` e a classe `LineWidthDialogFragment`. Para adicionar o layout de `fragment_line_width.xml`:

1. Expanda o nó `res/layout` do projeto no **Package Explorer**.
2. Clique com o botão direito do mouse na pasta `layout` e selecione **New > Android XML File** para exibir a caixa de diálogo **New Android XML File**.
3. No campo **File** da caixa de diálogo, digite `fragment_line_width.xml`.
4. Na seção **Root Element**, selecione `GridLayout` e, então, clique em **Finish**.
5. Na janela **Outline**, selecione o componente `GridLayout` e altere o valor de sua propriedade `id` para `@+id/LineWidthDialogGridLayout`.
6. Usando a **Palette** do editor **Graphical Layout**, arraste um componente `ImageView` e um componente `SeekBar` para o nó `LineWidthDialogGridLayout` na janela **Outline**, a fim de que a janela apareça como mostrado na Fig. 7.9. Configure a propriedade `id` de cada item como mostrado na figura.

7. Depois de terminar o passo 6, configure as propriedades dos componentes da interface gráfica do usuário com os valores mostrados na Fig. 7.10 e, então, salve e feche `fragment_line_width.xml`.

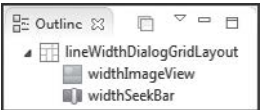


Figura 7.9 Visualização Outline para `fragment_line_width.xml`.

Componente da interface gráfica do usuário	Propriedade	Valor
lineWidthDialogGridLayout	Column Count	1
	Orientation	vertical
	Use Default Margins	true
widthImageView	Parâmetros do layout	
	Height	@dimen/line_imageview_height
	Gravity	fill_horizontal
	Outras propriedades	
widthSeekBar	Content Description	@string/line_imageview_description
	Parâmetros do layout	
	Gravity	fill_horizontal
	Outras propriedades	
	Max	50

Figura 7.10 Valores de propriedade para os componentes da interface gráfica do usuário em `fragment_line_width.xml`.

Adicionando a classe *LineWidthDialogFragment* ao projeto

Para adicionar a classe `LineWidthDialogFragment` ao projeto:

1. Clique com o botão direito do mouse no pacote `com.deitel.doodlz` na pasta `src` do projeto e selecione **New > Class** para exibir a caixa de diálogo **New Java Class**.
2. No campo **Name**, digite `LineWidthDialogFragment`.
3. No campo **Superclass**, mude a superclasse para `android.app.DialogFragment`.
4. Clique em **Finish** para criar a classe.

7.3.9 Adição da classe *EraseImageDialogFragment*

A classe `EraseImageDialogFragment` não exige um recurso de layout, pois vai exibir um componente `AlertDialog` simples contendo texto. Para adicionar a classe `EraseImageDialogFragment` ao projeto:

1. Clique com o botão direito do mouse no pacote `com.deitel.doodlz` na pasta `src` do projeto e selecione **New > Class** para exibir a caixa de diálogo **New Java Class**.
2. No campo **Name**, digite `EraseImageDialogFragment`.
3. No campo **Superclass**, mude a superclasse para `android.app.DialogFragment`.
4. Clique em **Finish** para criar a classe.

7.4 Classe MainActivity

O aplicativo consiste em seis classes:

- MainActivity (Fig. 7.11) – Serve como atividade pai para os fragmentos deste aplicativo.
- DoodleFragment (Seção 7.5) – Gerencia o componente DoodleView e o tratamento de eventos de acelerômetro.
- DoodleView (Seção 7.6) – Fornece os recursos de desenho, salvamento e impressão.
- ColorDialogFragment (Seção 7.7) – Um componente DialogFragment exibido quando o usuário dá um toque rápido em **COLOR** para definir a cor do desenho.
- LineWidthDialogFragment (Seção 7.8) – Um componente DialogFragment exibido quando o usuário dá um toque rápido em **LINE WIDTH** para definir a largura da linha.
- EraseImageDialogFragment (Seção 7.9) – Um componente DialogFragment exibido quando o usuário dá um toque rápido em **CLEAR** ou chacoalha o dispositivo para apagar o desenho atual.

O método onCreate da classe MainActivity (Fig. 7.11) infla a interface gráfica do usuário (linha 16) e, então, usa as técnicas que você aprendeu na Seção 5.2.2 para determinar o tamanho do dispositivo e configurar a orientação de MainActivity. Se este aplicativo estiver sendo executado em um dispositivo extragrande (linha 24), configuramos a orientação como paisagem (linhas 25 e 26); caso contrário, a configuramos como retrato (linhas 28 e 29).

```

1 // MainActivity.java
2 // Configura o layout de MainActivity
3 package com.deitel.doodlz;
4
5 import android.app.Activity;
6 import android.content.pm.ActivityInfo;
7 import android.content.res.Configuration;
8 import android.os.Bundle;
9
10 public class MainActivity extends Activity
11 {
12     @Override
13     protected void onCreate(Bundle savedInstanceState)
14     {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17
18         // determina o tamanho da tela
19         int screenSize =
20             getResources().getConfiguration().screenLayout &
21             Configuration.SCREENLAYOUT_SIZE_MASK;
22
23         // usa paisagem para tablets extragrandes; caso contrário, usa retrato
24         if (screenSize == Configuration.SCREENLAYOUT_SIZE_XLARGE)
25             setRequestedOrientation(
26                 ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
27         else

```

Figura 7.11 Classe MainActivity. (continua)

```

28         setRequestedOrientation(
29             ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
30     }
31 } // fim da classe MainActivity

```

Figura 7.11 Classe MainActivity.

7.5 Classe DoodleFragment

A classe DoodleFragment (Figs. 7.12 a 7.19) exibe o componente DoodleView (Seção 7.6), gerencia as opções de menu mostradas na barra de ação e no menu de opções e gerencia o tratamento de eventos de sensor para o recurso *chacoalhar para apagar* do aplicativo.

Instrução package, instruções import e campos

A Seção 7.2 discutiu as novas classes e interfaces importantes utilizadas pela classe DoodleFragment. Realçamos essas classes e interfaces na Figura 7.12. A variável `doodleView` de `DoodleView` (linha 22) representa a área de desenho. As variáveis `float` declaradas nas linhas 23 a 25 são usadas para calcular alterações na aceleração do dispositivo a fim de determinar quando ocorre um *evento de chacoalhar* (para que possamos perguntar se o usuário quer apagar o desenho), e a constante na linha 29 é usada para garantir que movimentos pequenos *não* sejam interpretados como um chacoalhar – escolhemos essa constante por tentativa e erro, chacoalhando o aplicativo em vários dispositivos. A linha 26 define uma variável `boolean` com o valor padrão `false`, a qual vai ser usada em toda essa classe para especificar quando uma caixa de diálogo está na tela a fim de que possamos impedir que várias caixas de diálogo sejam exibidas ao mesmo tempo – por exemplo, se a caixa de diálogo **Choose Color** está sendo exibida e o usuário chacoalha o dispositivo sem querer, a caixa de diálogo para apagar a imagem *não* deve aparecer.

```

1 // DoodleFragment.java
2 // Fragmento no qual o componente DoodleView é exibido
3 package com.deitel.doodlez;
4
5 import android.app.Fragment;
6 import android.content.Context;
7 import android.graphics.Color;
8 import android.hardware.Sensor;
9 import android.hardware.SensorEvent;
10 import android.hardware.SensorEventListener;
11 import android.hardware.SensorManager;
12 import android.os.Bundle;
13 import android.view.LayoutInflater;
14 import android.view.Menu;
15 import android.view.MenuInflater;
16 import android.view.MenuItem;
17 import android.view.View;
18 import android.view.ViewGroup;
19
20 public class DoodleFragment extends Fragment
21 {
22     private DoodleView doodleView; // trata eventos de toque e desenha

```

Figura 7.12 Instrução package, instruções import e campos da classe DoodleFragment. (continua)

```
23 private float acceleration;
24 private float currentAcceleration;
25 private float lastAcceleration;
26 private boolean dialogOnScreen = false;
27
28 // valor usado para determinar se o usuário chacoalhou o dispositivo para apagar
29 private static final int ACCELERATION_THRESHOLD = 100000;
30
```

Figura 7.12 Instrução package, instruções import e campos da classe DoodleFragment.

Sobrescrevendo o método onCreateView de Fragment

O método onCreateView (Fig. 7.13) infla a interface gráfica do usuário e inicializa as variáveis de instância de DoodleFragment. Assim como uma atividade, um fragmento pode colocar itens na barra de ação e no menu de opções do aplicativo. Para isso, o fragmento deve chamar seu método setHasOptionsMenu com o argumento true. Se a atividade pai também tem itens no menu de opções, então os itens da atividade e os do fragmento serão colocados na barra de ação e no menu de opções (de acordo com suas configurações).

```
31 // chamado quando a view do fragmento precisa ser criada
32 @Override
33 public View onCreateView(LayoutInflater inflater, ViewGroup container,
34 Bundle savedInstanceState)
35 {
36     super.onCreateView(inflater, container, savedInstanceState);
37     View view =
38         inflater.inflate(R.layout.fragment_doodle, container, false);
39
40     setHasOptionsMenu(true); // este fragmento tem itens de menu a exibir
41
42     // obtém referência para o componente DoodleView
43     doodleView = (DoodleView) view.findViewById(R.id.doodleView);
44
45     // inicializa valores de aceleração
46     acceleration = 0.00f;
47     currentAcceleration = SensorManager.GRAVITY_EARTH;
48     lastAcceleration = SensorManager.GRAVITY_EARTH;
49     return view;
50 }
51
```

Figura 7.13 Sobrescrevendo o método onCreateView de Fragment.

A linha 43 obtém uma referência para o componente DoodleView e, então, as linhas 46 a 48 inicializam as variáveis de instância que ajudam a calcular mudanças de aceleração para determinar se o usuário chacoalhou o dispositivo. Inicialmente, configuramos as variáveis currentAcceleration e lastAcceleration com a constante GRAVITY_EARTH de SensorManager, a qual representa a aceleração devido à gravidade da Terra. SensorManager também fornece constantes para outros planetas do sistema solar, para a Lua e para vários outros valores interessantes, os quais você pode ver no endereço:

<http://developer.android.com/reference/android/hardware/SensorManager.html>

Métodos `onStart` e `enableAccelerometerListening`

A *detecção do acelerômetro* deve ser habilitada somente quando o componente `DoodleFragment` está na tela. Por isso, sobrescrevemos o método de ciclo de vida `onStart` de `Fragment` (Fig. 7.14, linhas 53 a 58), o qual chama o método `enableAccelerometerListening` (linhas 61 a 72) para começar a detectar eventos de acelerômetro. Um componente `SensorManager` é usado para registrar receptores para eventos de acelerômetro.

```

52 // começa a detectar eventos de sensor
53 @Override
54 public void onStart()
55 {
56     super.onStart();
57     enableAccelerometerListening(); // detecta chacoalho
58 }
59
60 // habilita a detecção de eventos de acelerômetro
61 public void enableAccelerometerListening()
62 {
63     // obtém o componente SensorManager
64     SensorManager sensorManager =
65         (SensorManager) getActivity().getSystemService(
66             Context.SENSOR_SERVICE);
67
68     // registra a detecção de eventos de acelerômetro
69     sensorManager.registerListener(sensorEventListener,
70         sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
71         SensorManager.SENSOR_DELAY_NORMAL);
72 }
73

```

Figura 7.14 Métodos `onStart` e `enableAccelerometerListening`.

O método `enableAccelerometerListening` primeiramente usa `getSystemService` de `Activity` para recuperar o serviço `SensorManager` do sistema, o qual permite que o aplicativo interaja com os sensores do dispositivo. Então, as linhas 69 a 71 registram o recebimento de eventos de acelerômetro usando o método `registerListener` de `SensorManager`, o qual recebe três argumentos:

- O `SensorEventListener` que responde aos eventos (definidos na Fig. 7.16)
- Um objeto `Sensor` representando o tipo de dado de sensor que o aplicativo deseja receber – isso é recuperado chamando o método `getDefaultSensor` de `SensorManager` e passando uma constante do tipo `Sensor` (`Sensor.TYPE_ACCELEROMETER` neste aplicativo).
- Uma taxa na qual os eventos de sensor devem ser enviados para o aplicativo. Escolhemos `SENSOR_DELAY_NORMAL` para receber eventos de sensor na taxa padrão – uma taxa mais alta pode ser usada para obter dados mais precisos, mas isso também utiliza mais CPU e bateria.

Métodos `onPause` e `disableAccelerometerListening`

Para garantir que a *detecção do acelerômetro* seja desabilitada quando o elemento `DoodleFragment` não estiver na tela, sobrescrevemos o método de ciclo de vida `onPause` de `Fragment` (Fig. 7.15, linhas 75 a 80), o qual chama o método `disableAccelerometerListening` (linhas 83 a 93). O método `disableAccelerometerListening` usa o método `unregisterListener` da classe `SensorManager` para parar de detectar eventos de acelerômetro.

```

74 // para de detectar eventos de sensor
75 @Override
76 public void onPause()
77 {
78     super.onPause();
79     disableAccelerometerListening(); // para de detectar chacoalho
80 }
81
82 // desabilita a detecção de eventos de acelerômetro
83 public void disableAccelerometerListening()
84 {
85     // obtém o componente SensorManager
86     SensorManager sensorManager =
87         (SensorManager) getActivity().getSystemService(
88             Context.SENSOR_SERVICE);
89
90     // para de detectar eventos de acelerômetro
91     sensorManager.unregisterListener(sensorEventListener,
92         sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER));
93 }
94

```

Figura 7.15 Métodos onPause e disableAccelerometerListening.

Classe interna anônima que implementa a interface

SensorEventListener para processar eventos de acelerômetro

A Figura 7.16 sobrescreve o método **onSensorChanged** de **SensorEventListener** (linhas 100 a 125) para processar eventos de acelerômetro. Se o usuário movimenta o dispositivo, esse método determina se o movimento foi suficiente para ser considerado um chacoalhar. Em caso positivo, a linha 123 chama o método **confirmErase** (Fig. 7.17) para exibir um elemento **EraseImageDialogFragment** (Seção 7.9) e confirmar se o usuário realmente quer apagar a imagem. A interface **SensorEventListener** contém também o método **onAccuracyChanged** (linhas 128 a 131) – não usamos esse método neste aplicativo, de modo que fornecemos um corpo vazio, pois o método é exigido pela interface.

```

95 // rotina de tratamento para eventos de acelerômetro
96 private SensorEventListener sensorEventListener =
97     new SensorEventListener()
98     {
99         // usa acelerômetro para determinar se o usuário chacoalhou o dispositivo
100         @Override
101         public void onSensorChanged(SensorEvent event)
102         {
103             // garante que outras caixas de diálogo não sejam exibidas
104             if (!dialogOnScreen)
105             {
106                 // obtém valores x, y e z para o componente SensorEvent
107                 float x = event.values[0];
108                 float y = event.values[1];
109                 float z = event.values[2];
110
111                 // salva valor de aceleração anterior
112                 lastAcceleration = currentAcceleration;
113
114                 // calcula a aceleração atual

```

Figura 7.16 Classe interna anônima que implementa a interface **SensorEventListener** para processar eventos de acelerômetro.

```

115         currentAcceleration = x * x + y * y + z * z;
116
117         // calcula a mudança na aceleração
118         acceleration = currentAcceleration *
119             (currentAcceleration - lastAcceleration);
120
121         // se a aceleração está acima de determinado limite
122         if (acceleration > ACCELERATION_THRESHOLD)
123             confirmErase();
124     }
125 } // fim do método onSensorChanged
126
127 // método obrigatório da interface SensorEventListener
128 @Override
129 public void onAccuracyChanged(Sensor sensor, int accuracy)
130 {
131 }
132 }; // fim da classe interna anônima
133

```

Figura 7.16 Classe interna anônima que implementa a interface `SensorEventListener` para processar eventos de acelerômetro.

O usuário pode chacoalhar o dispositivo mesmo quando caixas de diálogo já estão exibidas na tela. Por isso, `onSensorChanged` primeiramente verifica se uma caixa de diálogo está sendo exibida (linha 104). Esse teste garante que nenhuma outra caixa de diálogo esteja sendo exibida; caso contrário, `onSensorChanged` simplesmente retorna. Isso é importante, pois os eventos de sensor ocorrem em uma thread de execução diferente. Sem esse teste, poderíamos exibir a caixa de diálogo de confirmação para apagar a imagem quando outra caixa de diálogo estivesse na tela.

O parâmetro **SensorEvent** contém informações sobre a alteração ocorrida no sensor. Para eventos de acelerômetro, o array `values` desse parâmetro contém três elementos, representando a aceleração (em *metros/segundo*²) nas direções *x* (esquerda/direita), *y* (para cima/para baixo) e *z* (para frente/para trás). Uma descrição e o diagrama do sistema de coordenadas usado pela API de `SensorEvent` estão disponíveis no endereço:

developer.android.com/reference/android/hardware/SensorEvent.html

Esse link também descreve os significados no mundo real dos valores de *x*, *y* e *z* de `SensorEvent` para cada objeto `Sensor` diferente.

As linhas 107 a 109 armazenam os valores de aceleração. É importante tratar os eventos de sensor rapidamente ou copiar os dados do evento (como fizemos), pois o array de valores de sensor é *reutilizado* para cada evento de sensor. A linha 112 armazena o último valor de `currentAcceleration`. A linha 115 soma os quadrados dos valores de *x*, *y* e *z* da aceleração e os armazena em `currentAcceleration`. Em seguida, usando os valores de `currentAcceleration` e `lastAcceleration`, calculamos um valor (`acceleration`) que pode ser comparado com nossa constante `ACCELERATION_THRESHOLD`. Se o valor for maior que a constante, o usuário movimentou o dispositivo o bastante para que o aplicativo considere o movimento um chacoalhar. Nesse caso, chamamos o método `confirmErase`.

Método `confirmErase`

O método `confirmErase` (Fig. 7.17) simplesmente cria um elemento `EraseImageDialogFragment` (Seção 7.9) e utiliza o método `show` de `DialogFragment` para exibi-lo.


```

134 // confirma se a imagem deve ser apagada
135 private void confirmErase()
136 {
137     EraseImageDialogFragment fragment = new EraseImageDialogFragment();
138     fragment.show(getFragmentManager(), "erase dialog");
139 }
140

```

Figura 7.17 O método `confirmErase` exibe um elemento `EraseImageDialogFragment`.

Métodos sobrescritos `onCreateOptionsMenu` e `onOptionsItemSelected` de `Fragment`

A Figura 7.18 sobrescreve o método `onCreateOptionsMenu` de `Fragment` (linhas 142 a 147) para adicionar as opções ao argumento `Menu` do método usando seu argumento `MenuInflater`. Quando o usuário seleciona um item de menu, o método `onOptionsItemSelected` de `Fragment` (linhas 150 a 180) responde à seleção.

```

141 // exibe os itens de menu deste fragmento
142 @Override
143 public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
144 {
145     super.onCreateOptionsMenu(menu, inflater);
146     inflater.inflate(R.menu.doodle_fragment_menu, menu);
147 }
148
149 // trata a escolha no menu de opções
150 @Override
151 public boolean onOptionsItemSelected(MenuItem item)
152 {
153     // escolhe com base no identificador de MenuItem
154     switch (item.getItemId())
155     {
156         case R.id.color:
157             ColorDialogFragment colorDialog = new ColorDialogFragment();
158             colorDialog.show(getFragmentManager(), "color dialog");
159             return true; // consome o evento de menu
160         case R.id.lineWidth:
161             LineWidthDialogFragment widthDialog =
162                 new LineWidthDialogFragment();
163             widthDialog.show(getFragmentManager(), "line width dialog");
164             return true; // consome o evento de menu
165         case R.id.eraser:
166             doodleView.setDrawingColor(Color.WHITE); // cor de linha branca
167             return true; // consome o evento de menu
168         case R.id.clear:
169             confirmErase(); // confirma antes de apagar a imagem
170             return true; // consome o evento de menu
171         case R.id.save:
172             doodleView.saveImage(); // salva a imagem atual
173             return true; // consome o evento de menu
174         case R.id.print:
175             doodleView.printImage(); // imprime a imagem atual
176             return true; // consome o evento de menu
177     } // fim de switch
178
179     return super.onOptionsItemSelected(item); // chama o método de super
180 } // fim do método onOptionsItemSelected
181

```

Figura 7.18 Métodos sobrescritos `onCreateOptionsMenu` e `onOptionsItemSelected` de `Fragment`.

Usamos o método `getItemID` do argumento `MenuItem` (linha 154) para obter o identificador de recurso do item de menu selecionado e, então, adotamos diferentes ações, de acordo com a seleção. As ações são as seguintes:

- Para `R.id.color`, as linhas 157 e 158 criam e mostram um elemento `ColorDialogFragment` (Seção 7.7) para permitir que o usuário selecione uma nova cor de desenho.
- Para `R.id.linewidth`, as linhas 161 a 163 criam e mostram um elemento `LineWidthDialogFragment` (Seção 7.8) para permitir que o usuário selecione uma nova espessura de linha.
- Para `R.id.eraser`, a linha 166 configura a cor de desenho de `doodleView` como branca, o que transforma os dedos do usuário em *borrachas*.
- Para `R.id.clear`, a linha 169 chama o método `confirmErase` (Fig. 7.17) para exibir um elemento `EraseImageDialogFragment` (Seção 7.9) e confirmar se o usuário realmente quer apagar a imagem.
- Para `R.id.save`, a linha 172 chama o método `saveImage` de `doodleView` para salvar a pintura como uma imagem armazenada na *Galeria* do dispositivo.
- Para `R.id.print`, a linha 175 chama o método `printImage` de `doodleView` para permitir que o usuário salve a imagem como PDF ou a imprima.

Métodos `getDoodleView` e `setDialogOnScreen`

Os métodos `getDoodleView` e `setDialogOnScreen` (Fig. 7.19) são chamados pelos métodos das subclasses de `DialogFragment` do aplicativo. O método `getDoodleView` retorna uma referência para o objeto `DoodleView` desse fragmento para que um elemento `DialogFragment` possa definir a cor de desenho, a largura da linha ou limpar a imagem. O método `setDialogOnScreen` é chamado pelos métodos de ciclo de vida de `Fragment` das subclasses de `DialogFragment` do aplicativo, para indicar quando uma caixa de diálogo está na tela.

```

182 // retorna o objeto DoodleView
183 public DoodleView getDoodleView()
184 {
185     return doodleView;
186 }
187
188 // indica se uma caixa de diálogo está sendo exibida
189 public void setDialogOnScreen(boolean visible)
190 {
191     dialogOnScreen = visible;
192 }
193 }
```

Figura 7.19 Métodos `getDoodleView` e `setDialogOnScreen`.

7.6 Classe `DoodleView`

A classe `DoodleView` (Figs. 7.20 a 7.33) processa os toques do usuário e desenha as linhas correspondentes.

A instrução `package` e as instruções `import` de `DoodleView`

A Figura 7.20 lista a instrução `package`, as instruções `import` e os campos da classe `DoodleView`. As classes e interfaces novas estão realçadas aqui. Muitas delas foram discutidas na

Seção 7.2 e as restantes serão discutidas na medida em que as usarmos ao longo da classe `DoodleView`.

```

1  // DoodleView.java
2  // Visualização principal do aplicativo Doodlz.
3  package com.deitel.doodlz;
4
5  import java.util.HashMap;
6  import java.util.Map;
7
8  import android.content.Context;
9  import android.graphics.Bitmap;
10 import android.graphics.Canvas;
11 import android.graphics.Color;
12 import android.graphics.Paint;
13 import android.graphics.Path;
14 import android.graphics.Point;
15 import android.os.Build;
16 import android.provider.MediaStore;
17 import android.support.v4.print.PrintHelper;
18 import android.util.AttributeSet;
19 import android.view.GestureDetector;
20 import android.view.GestureDetector.SimpleOnGestureListener;
21 import android.view.Gravity;
22 import android.view.MotionEvent;
23 import android.view.View;
24 import android.widget.Toast;
25

```

Figura 7.20 Instrução package e instruções import de `DoodleView`.

Variáveis estáticas e variáveis de instância de `DoodleView`

As variáveis estáticas e de instância da classe `DoodleView` (Fig. 7.21, linhas 30 a 43) são usadas para gerenciar os dados do conjunto de linhas que o usuário está desenhando e para desenhar essas linhas. A linha 38 cria o elemento `PathMap`, o qual mapeia cada identificador de dedo (conhecido como ponteiro) em um objeto `Path` correspondente para as linhas que estão sendo desenhadas. As linhas 39 e 40 criam o objeto `previousPointMap`, o qual mantém o último ponto de cada dedo – à medida que cada dedo se move, desenharmos uma linha de seu ponto atual até seu ponto anterior. Vamos discutir outros campos à medida que os utilizarmos na classe `DoodleView`.

```

26 // a tela principal que é pintada
27 public class DoodleView extends View
28 {
29     // usado para determinar se o usuário moveu um dedo o suficiente para
    // desenhar novamente
30     private static final float TOUCH_TOLERANCE = 10;
31
32     private Bitmap bitmap; // área de desenho para exibir ou salvar
33     private Canvas bitmapCanvas; // usado para desenhar no bitmap
34     private final Paint paintScreen; // usado para desenhar o bitmap na tela
35     private final Paint paintLine; // usado para desenhar linhas no bitmap
36
37     // mapas dos objetos Path que estão sendo desenhados e os
    // objetos Point desses objetos Path
38     private final Map<Integer, Path> pathMap = new HashMap<Integer, Path>();

```

Figura 7.21 Variáveis estáticas e variáveis de instância de `DoodleView`.

```

39     private final Map<Integer, Point> previousPointMap =
40         new HashMap<Integer, Point>();
41
42     // usado para ocultar/mostrar barras de sistema
43     private GestureDetector singleTapDetector;
44

```

Figura 7.21 Variáveis estáticas e variáveis de instância de DoodleView.

Construtor de DoodleView

O construtor (Fig. 7.22) inicializa diversas variáveis de instância da classe – os dois componentes Map são inicializados em suas declarações na Fig. 7.21. A linha 49 cria o objeto PaintScreen de Paint que vai ser usado para exibir o desenho do usuário na tela, e a linha 52 cria o objeto PaintLine que especifica as configurações da(s) linha(s) que o usuário está desenhando. As linhas 53 a 57 especificam as configurações do objeto paintLine. Passamos true para o método **setAntiAlias** de Paint para habilitar o *anti-aliasing* que suaviza as bordas das linhas. Em seguida, configuramos o estilo de Paint como Paint.Style.STROKE, com o método **setStyle** de Paint. O estilo pode ser STROKE, FILL ou FILL_AND_STROKE para uma linha, para uma forma preenchida sem borda e para uma forma preenchida com borda, respectivamente. A opção padrão é Paint.Style.FILL. Configuramos a largura da linha usando o método **setStrokeWidth** de Paint. Isso define a *largura de linha padrão* do aplicativo como cinco pixels. Também usamos o método **setStrokeCap** de Paint para arredondar as extremidades das linhas com Paint.Cap.ROUND. As linhas 60 e 61 criam um objeto GestureDetector que utiliza o elemento singleTapListener para buscar eventos de toque rápido.

```

45 // o construtor de DoodleView o inicializa
46 public DoodleView(Context context, AttributeSet attrs)
47 {
48     super(context, attrs); // passa o contexto para o construtor de View
49     paintScreen = new Paint(); // usado para exibir bitmap na tela
50
51     // ajusta as configurações de exibição iniciais da linha pintada
52     paintLine = new Paint();
53     paintLine.setAntiAlias(true); // suaviza as bordas da linha desenhada
54     paintLine.setColor(Color.BLACK); // a cor padrão é preto
55     paintLine.setStyle(Paint.Style.STROKE); // linha cheia
56     paintLine.setStrokeWidth(5); // configura a largura de linha padrão
57     paintLine.setStrokeCap(Paint.Cap.ROUND); // extremidades da linha arredondadas
58
59     // GestureDetector para toques rápidos
60     singleTapDetector =
61         new GestureDetector(getContext(), singleTapListener);
62 }
63

```

Figura 7.22 Construtor de DoodleView.

Método sobrescrito onSizeChanged de View

O tamanho de DoodleView não é determinado até que ele seja inflado e adicionado à hierarquia de Views de MainActivity; portanto, não podemos determinar o tamanho do objeto Bitmap de desenho em onCreate. Assim, sobrescrevemos o método onSizeChanged de View (Fig. 7.23), o qual é chamado quando o tamanho de DoodleView é modificado – por exemplo, quando é adicionado à hierarquia de Views de uma atividade ou quando o

usuário gira o dispositivo. Neste aplicativo, `onSizeChanged` é chamado somente quando `DoodleView` é adicionado à hierarquia de Views da atividade de `Doodlez`, pois o aplicativo sempre aparece no modo *retrato* em telefones e tablets pequenos, e no modo *paisagem* em tablets grandes.

```

64 // O método onSizeChanged cria Bitmap e Canvas após exibir o aplicativo
65 @Override
66 public void onSizeChanged(int w, int h, int oldW, int oldH)
67 {
68     bitmap = Bitmap.createBitmap(getWidth(), getHeight(),
69     Bitmap.Config.ARGB_8888);
70     bitmapCanvas = new Canvas(bitmap);
71     bitmap.eraseColor(Color.WHITE); // apaga o Bitmap com branco
72 }
73

```

Figura 7.23 Método sobrescrito `onSizeChanged` de View.

O método estático `createBitmap` de `Bitmap` cria um objeto `Bitmap` com a largura e a altura especificadas – aqui, usamos a largura e a altura de `DoodleView` como dimensões do `Bitmap`. O último argumento de `createBitmap` é a codificação de `Bitmap`, a qual especifica como cada pixel do objeto `Bitmap` é armazenado. A constante `Bitmap.Config.ARGB_8888` indica que a cor de cada pixel é armazenada em quatro bytes (um byte para cada valor de alfa, vermelho, verde e azul) da cor do pixel. Em seguida, criamos um novo `Canvas`, que é usado para desenhar formas diretamente no objeto `Bitmap`. Por fim, usamos o método `eraseColor` de `Bitmap` para preencher o objeto `Bitmap` com pixels brancos – o fundo padrão de `Bitmap` é preto.

Métodos `clear`, `setDrawingColor`, `getDrawingColor`, `setLineWidth` e `getLineWidth` de `DoodleView`

A Figura 7.24 define os métodos `clear` (linhas 75 a 81), `setDrawingColor` (linhas 84 a 87), `getDrawingColor` (linhas 90 a 93), `setLineWidth` (linhas 96 a 99) e `getLineWidth` (linhas 102 a 105), os quais são chamados a partir de `DoodleFragment`. O método `clear`, o qual usamos no elemento `EraseImageDialogFragment`, esvazia `PathMap` e `previousPointMap`, apaga o objeto `Bitmap` configurando todos os seus pixels com a cor branca e, então, chama o método `invalidate` herdado de `View` para indicar que a `View` precisa ser redeseenhada. Em seguida, o sistema determina automaticamente quando o método `onDraw` de `View` deve ser chamado. O método `setDrawingColor` muda a cor de desenho atual, configurando a cor do objeto `paintLine` de `Paint`. O método `setColor` de `Paint` recebe um valor `int` representando a nova cor no formato `ARGB`. O método `getDrawingColor` retorna a cor atual, a qual usamos em `ColorDialogFragment`. O método `setLineWidth` configura a largura do traço de `paintLine` com o número especificado de pixels. O método `getLineWidth` retorna a largura de traço atual, a qual usamos em `LineWidthDialogFragment`.

```

74 // limpa o desenho
75 public void clear()
76 {
77     pathMap.clear(); // remove todos os caminhos
78     previousPointMap.clear(); // remove todos os pontos anteriores

```

Figura 7.24 Métodos `clear`, `setDrawingColor`, `getDrawingColor`, `setLineWidth` e `getLineWidth` de `DoodleView`.

```

79     bitmap.eraseColor(Color.WHITE); // apaga o bitmap
80     invalidate(); // atualiza a tela
81 }
82
83 // configura a cor da linha pintada
84 public void setDrawingColor(int color)
85 {
86     paintLine.setColor(color);
87 }
88
89 // retorna a cor da linha pintada
90 public int getDrawingColor()
91 {
92     return paintLine.getColor();
93 }
94
95 // configura a largura da linha pintada
96 public void setLineWidth(int width)
97 {
98     paintLine.setStrokeWidth(width);
99 }
100
101 // retorna a largura da linha pintada
102 public int getLineWidth()
103 {
104     return (int) paintLine.setStrokeWidth();
105 }
106

```

Figura 7.24 Métodos `clear`, `setDrawingColor`, `getDrawingColor`, `setLineWidth` e `getLineWidth` de `DoodleView`.

Método sobrescrito `onDraw` de `View`

Quando uma `View` precisa ser *redesenhada*, seu método `onDraw` é chamado. A Figura 7.25 sobrescreve `onDraw` para exibir o `bitmap` (o objeto `Bitmap` que contém o desenho) em `DoodleView`, chamando o método `drawBitmap` do argumento `Canvas`. O primeiro argumento é o objeto `Bitmap` a ser desenhado, os dois argumentos seguintes são as coordenadas *x-y* onde deve ser colocado o canto superior esquerdo do objeto `Bitmap` na `View`, e o último argumento é o objeto `Paint` que especifica as características do desenho. Então, as linhas 115 e 116 fazem um loop pelos componentes `Path` que estão sendo desenhados e os exibem. Para cada chave `Integer` no elemento `pathMap`, passamos o objeto `Path` correspondente para o método `drawPath` de `Canvas`, para desenhar o objeto `Path` usando o objeto `paintLine`, o qual define a *largura* e a *cor* da linha.

```

107 // chamado sempre que essa View é desenhada
108 @Override
109 protected void onDraw(Canvas canvas)
110 {
111     // desenha a tela de fundo
112     canvas.drawBitmap(bitmap, 0, 0, paintScreen);
113
114     // para cada caminho que está sendo desenhado
115     for (Integer key : pathMap.keySet())
116         canvas.drawPath(pathMap.get(key), paintLine); // desenha a linha
117 }
118

```

Figura 7.25 Método sobrescrito `onDraw` de `View`.

Métodos `hideSystemBars` e `showSystemBars` de `DoodleView`

Este aplicativo utiliza o novo *modo imersivo* do Android 4.4 para permitir que os usuários desenhem na tela inteira. Quando o usuário dá um toque rápido na tela, um elemento `SimplyOnGestureListener` de `GestureDetector` (Fig. 7.27) determina se as barras de sistema e a barra de ação estão sendo exibidas. Em caso positivo, o método `hideSystemBars` (Fig. 7.26, linhas 120 a 130) é chamado; caso contrário, o método `showSystemBars` (Fig. 7.26, linhas 133 a 140) é chamado. Para este aplicativo, habilitamos o modo imersivo somente para Android 4.4. Assim, ambos os métodos primeiro verificam se a versão de Android em execução no dispositivo – `Build.VERSION.SDK_INT` – é maior ou igual à constante do Android 4.4 (API nível 19) – `Build.VERSION_CODES.KITKAT`. Se for, os dois métodos usam o método `setSystemUiVisibility` de `View` para configurar as barras de sistema e a barra de ação. Para ocultar as barras de sistema e a barra de ação e colocar a interface do usuário no modo imersivo, passamos para `setSystemUiVisibility` as constantes que são combinadas por meio do operador OU bit a bit (`|`) nas linhas 124 a 129. Para mostrar as barras de sistema e a barra de ação, passamos para `setSystemUiVisibility` as constantes que são combinadas nas linhas 137 a 139. Essas combinações de constantes de `View` garantem que o elemento `DoodleView` não seja redimensionado toda vez que as barras de sistema e a barra de ação forem ocultadas e reexibidas. Em vez disso, as barras de sistema e a barra de ação se *sobrepoem* ao elemento `DoodleView` – isto é, parte do elemento `DoodleView` fica temporariamente oculta quando as barras de sistema e a barra de ação estão na tela. A constante `View.SYSTEM_UI_FLAG_IMMERSIVE` é novidade no Android 4.4. Para obter mais informações sobre o modo imersivo, visite:

<http://developer.android.com/training/system-ui/immersive.html>

```

119 // oculta as barras de sistema e a barra de ação
120 public void hideSystemBars()
121 {
122     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT)
123         setSystemUiVisibility(
124             View.SYSTEM_UI_FLAG_LAYOUT_STABLE |
125             View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION |
126             View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN |
127             View.SYSTEM_UI_FLAG_HIDE_NAVIGATION |
128             View.SYSTEM_UI_FLAG_FULLSCREEN |
129             View.SYSTEM_UI_FLAG_IMMERSIVE);
130 }
131
132 // mostra as barras de sistema e a barra de ação
133 public void showSystemBars()
134 {
135     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT)
136         setSystemUiVisibility(
137             View.SYSTEM_UI_FLAG_LAYOUT_STABLE |
138             View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION |
139             View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN);
140 }
141
```

Figura 7.26 Métodos `hideSystemBars` e `showSystemBars` de `DoodleView`.

Classe interna anônima que implementa a interface SimpleOnGestureListener

A Figura 7.27 cria o receptor SimpleOnGestureListener chamado singleTapListener, o qual foi registrado nas linhas 60 e 61 (Fig. 7.22) com GestureDetector. Lembre-se de que SimpleOnGestureListener é uma classe adaptadora que implementa as interfaces OnGestureListener e OnDoubleTapListener. Os métodos simplesmente retornam false – indicando que os eventos *não foram tratados*. Sobrescrevemos somente o método **onSingleTap** (linhas 146 a 155), o qual é chamado quando o usuário dá toques rápidos na tela. Determinamos se as barras de sistema e a barra de aplicativo estão sendo exibidas (linhas 149 e 150) chamando o método **getSystemUiVisibility** de View e combinando seu resultado com a constante View.SYSTEM_UI_FLAG_HIDE_NAVIGATION. Se o resultado for 0, as barras de sistema e a barra de aplicativo estão sendo exibidas; portanto, chamamos o método **hideSystemBars**; caso contrário, chamamos **showSystemBars**. Retornar true indica que o evento de toque rápido foi tratado.

```

142 // cria SimpleOnGestureListener para eventos de toque rápido
143 private SimpleOnGestureListener singleTapListener =
144     new SimpleOnGestureListener()
145     {
146         @Override
147         public boolean onSingleTapUp(MotionEvent e)
148         {
149             if ((getSystemUiVisibility() &
150                 View.SYSTEM_UI_FLAG_HIDE_NAVIGATION) == 0)
151                 hideSystemBars();
152             else
153                 showSystemBars();
154             return true;
155         }
156     };
157 
```

Figura 7.27 Classe interna anônima que implementa a interface SimpleOnGestureListener.

Método sobrescrito onTouchEvent de View

O método onTouchEvent (Fig. 7.28) é chamado quando a View recebe um evento de toque. O Android suporta *multitouch* – isto é, vários dedos tocando na tela. A qualquer momento, o usuário pode tocar na tela com mais dedos ou retirar os dedos dela. Por isso, cada dedo – conhecido como *ponteiro* – tem um identificador exclusivo que o diferencia nos eventos de toque. Vamos usar esse identificador para localizar os objetos Path correspondentes que representam cada linha que está sendo desenhada. Esses objetos Path são armazenados em pathMap.

```

158 // trata evento de toque
159 @Override
160 public boolean onTouchEvent(MotionEvent event)
161 {

```

Figura 7.28 Método sobrescrito onTouchEvent de View. (continua)


```

162 // obtém o tipo de evento e o identificador do ponteiro que causou o evento
163 // se um evento de toque rápido ocorreu em dispositivo KitKat ou mais recente
164 if (singleTapDetector.onTouchEvent(event))
165     return true;
166
167 int action = event.getActionMasked(); // tipo de evento
168 int actionIndex = event.getActionIndex(); // ponteiro (isto é, o dedo)
169
170 // determina se o toque começou, terminou ou está ocorrendo
171 if (action == MotionEvent.ACTION_DOWN ||
172     action == MotionEvent.ACTION_POINTER_DOWN)
173 {
174     touchStarted(event.getX(actionIndex), event.getY(actionIndex),
175                 event.getPointerId(actionIndex));
176 }
177 else if (action == MotionEvent.ACTION_UP ||
178          action == MotionEvent.ACTION_POINTER_UP)
179 {
180     touchEnded(event.getPointerId(actionIndex));
181 }
182 else
183 {
184     touchMoved(event);
185 }
186
187 invalidate(); // redesenha
188 return true;
189 } // fim do método onTouchEvent
190

```

Figura 7.28 Método sobrescrito onTouchEvent de View.

Quando ocorre um evento de toque, a linha 164 chama o método onTouchEvent de GestureDetector (singleTapDetector) para primeiramente determinar se o evento foi um toque rápido, a fim de ocultar ou mostrar as barras de sistema e a barra de aplicativo. Se o evento de movimento foi um toque rápido, o método retorna imediatamente.

O método **getActionMasked** de MotionEvent (linha 167) retorna um valor int representando o tipo de MotionEvent, o qual você pode usar com constantes da classe MotionEvent para determinar como vai tratar cada evento. O método **getActionIndex** de MotionEvent (linha 168) retorna um índice inteiro representando o dedo que causou o evento. Esse índice *não* é o identificador exclusivo do dedo – é apenas o índice no qual a informação do dedo está localizada nesse objeto MotionEvent. Para obtermos o identificador exclusivo do dedo, que persiste entre os objetos MotionEvent até que o usuário retire esse dedo da tela, vamos usar o método **getPointerID** de MotionEvent (linhas 175 e 180), passando o índice do dedo como argumento.

Se a ação for MotionEvent.ACTION_DOWN ou MotionEvent.ACTION_POINTER_DOWN (linhas 171 e 172), o usuário *tocou na tela com um novo dedo*. O primeiro dedo a tocar na tela gera um evento MotionEvent.ACTION_DOWN, e todos os outros dedos geram eventos MotionEvent.ACTION_POINTER_DOWN. Para esses casos, chamamos o método touchStarted (Fig. 7.29) para armazenar as coordenadas iniciais do toque. Se a ação for MotionEvent.ACTION_UP ou MotionEvent.ACTION_POINTER_UP, o usuário *retirou um dedo da tela*, de modo que chamamos o método touchEnded (Fig. 7.31) para desenhar o objeto Path completo no bitmap, a fim de que tenhamos um registro permanente desse objeto Path. Para todos os outros eventos de toque, chamamos o método touchMoved (Fig. 7.30) para desenhar as linhas. Depois que o evento é processado, a linha 187 chama o método **invalidate** herdado de View para redesenhar a tela, e a linha 188 retorna true para indicar que o evento foi processado.

Método *touchStarted* da classe *DoodleView*

O método *touchStarted* (Fig. 7.29) é chamado quando um dedo *toca* a tela pela primeira vez. As coordenadas do toque e seu identificador são fornecidos como argumentos. Se um objeto *Path* já existe para determinado identificador (linha 198), chamamos o método **reset** de *Path* para *limpar* quaisquer pontos já existentes, a fim de que possamos *reutilizar* o objeto *Path* para um novo traço. Caso contrário, criamos um novo objeto *Path*, o adicionamos a *PathMap* e, então, adicionamos um novo objeto *Point* a *previousPointMap*. As linhas 213 a 215 chamam o método **moveTo** de *Path* para configurar as coordenadas iniciais de *Path* e especificar os novos valores *x* e *y* de *Point*.

```

191 // chamado quando o usuário toca na tela
192 private void touchStarted(float x, float y, int lineID)
193 {
194     Path path; // usado para armazenar o caminho para o identificador de toque dado
195     Point point; // usado para armazenar o último ponto no caminho
196
197     // se já existe um caminho para lineID
198     if (pathMap.containsKey(lineID))
199     {
200         path = pathMap.get(lineID); // obtém o objeto Path
201         path.reset(); // redefine o objeto Path, pois um novo toque começou
202         point = previousPointMap.get(lineID); // obtém o último ponto de Path
203     }
204     else
205     {
206         path = new Path();
207         pathMap.put(lineID, path); // adiciona o objeto Path ao mapa
208         point = new Point(); // cria um novo objeto Point
209         previousPointMap.put(lineID, point); // adiciona o objeto Point ao mapa
210     }
211
212     // move até as coordenadas do toque
213     path.moveTo(x, y);
214     point.x = (int) x;
215     point.y = (int) y;
216 } // fim do método touchStarted
217

```

Figura 7.29 Método *touchStarted* da classe *DoodleView*.

Método *touchMoved* da classe *DoodleView*

O método *touchMoved* (Fig. 7.30) é chamado quando o usuário movimenta um ou mais dedos pela tela. O evento de sistema *MotionEvent* passado a partir de *onTouchEvent* contém informações de toque para vários movimentos na tela, caso ocorram ao mesmo tempo. O método **getPointerCount** de *MotionEvent* (linha 222) retorna o número de toques que esse objeto *MotionEvent* descreve. Para cada um, armazenamos o identificador do dedo (linha 225) em *pointerID* e armazenamos o índice do dedo correspondente desse objeto *MotionEvent* (linha 226) em *pointerIndex*. Então, verificamos se existe um objeto *Path* correspondente no *HashMap pathMap* (linha 229). Se existir, usamos os métodos *getX* e *getY* de *MotionEvent* para obter as últimas coordenadas desse evento de *arrastar* para o *pointerIndex* especificado. Obtemos o objeto *Path* correspondente e o último objeto *Point* para o *pointerID* de cada *HashMap* respectivo e, então, calculamos a diferença entre o último ponto e o ponto atual – queremos atualizar o objeto *Path* *somente* se o usuário tiver realizado um movimento por uma distância maior que nossa constante *TOUCH_TOLERANCE*. Fazemos isso porque muitos dispositivos são sensíveis o suficiente para gerar objetos *MotionEvent*

indicando pequenos movimentos quando o usuário está tentando manter um dedo imóvel na tela. Se o usuário movimentou um dedo por mais do que `TOUCH_TOLERANCE`, usamos o método `quadTo` de `Path` (linhas 248 e 249) para adicionar uma curva geométrica (especificamente, uma *curva Bezier quadrática*) do objeto `Point` anterior até o novo objeto `Point`. Então, atualizamos o objeto `Point` mais recente para esse dedo.

```

218 // chamado quando o usuário arrasta o dedo pela tela
219 private void touchMoved(MotionEvent event)
220 {
221     // para cada um dos ponteiros em MotionEvent
222     for (int i = 0; i < event.getPointerCount(); i++)
223     {
224         // obtém o identificador e o índice do ponteiro
225         int pointerID = event.getPointerId(i);
226         int pointerIndex = event.findPointerIndex(pointerID);
227
228         // se existe um caminho associado ao ponteiro
229         if (pathMap.containsKey(pointerID))
230         {
231             // obtém as novas coordenadas do ponteiro
232             float newX = event.getX(pointerIndex);
233             float newY = event.getY(pointerIndex);
234
235             // obtém o objeto Path e o objeto Point
236             // anterior associados a esse ponteiro
237             Path path = pathMap.get(pointerID);
238             Point point = previousPointMap.get(pointerID);
239
240             // calcula quanto o usuário moveu a partir da última atualização
241             float deltaX = Math.abs(newX - point.x);
242             float deltaY = Math.abs(newY - point.y);
243
244             // se a distância é significativa o suficiente para ter importância
245             if (deltaX >= TOUCH_TOLERANCE || deltaY >= TOUCH_TOLERANCE)
246             {
247                 // move o caminho para o novo local
248                 path.quadTo(point.x, point.y, (newX + point.x) / 2,
249                     (newY + point.y) / 2);
250
251                 // armazena as novas coordenadas
252                 point.x = (int) newX;
253                 point.y = (int) newY;
254             }
255         }
256     }
257 } // fim do método touchMoved
258

```

Figura 7.30 Método `touchMoved` da classe `DoodleView`.

Método `touchEnded` da classe `DoodleView`

O método `touchEnded` (Fig. 7.31) é chamado quando o usuário retira um dedo da tela. O método recebe como argumento o identificador do dedo (`lineID`) para o toque que acabou de terminar. A linha 262 obtém o objeto `Path` correspondente. A linha 263 chama o método `drawPath` de `bitmapCanvas` para desenhar o `Path` no objeto `Bitmap` chamado `bitmap` antes de chamarmos o método `reset` de `Path` para limpar o traço. Redefinir `Path` não apaga sua linha pintada correspondente da tela, pois essas linhas já foram desenhadas no `bitmap` que está sendo exibido. As linhas que estão sendo desenhadas pelo usuário são exibidas por cima desse `bitmap`.

```

259 // chamado quando o usuário finaliza um toque
260 private void touchEnded(int lineID)
261 {
262     Path path = pathMap.get(lineID); // obtém o objeto Path correspondente
263     bitmapCanvas.drawPath(path, paintLine); // desenha em bitmapCanvas
264     path.reset(); // redefine o objeto Path
265 }
266

```

Figura 7.31 Método touchEnded da classe DoodleView.

Método saveImage de DoodleView

O método saveImage (Fig. 7.32) salva o desenho atual em um arquivo na galeria do dispositivo. A linha 271 cria um nome de arquivo para a imagem e, então, as linhas 274 a 276 armazenam a imagem na Galeria do dispositivo chamando o método insertImage da classe MediaStore.Images.Media. O método recebe quatro argumentos:

- um ContentResolver, utilizado para determinar onde a imagem deve ser armazenada no dispositivo
- o objeto Bitmap a ser armazenado
- o nome da imagem
- uma descrição da imagem

O método insertImage retorna um objeto String representando o local da imagem no dispositivo, ou null, caso a imagem não possa ser salva. As linhas 278 a 295 verificam se a imagem foi salva e exibem o elemento Toast apropriado.

```

267 // salva a imagem atual na Galeria
268 public void saveImage()
269 {
270     // usa "Doodlz", seguido da hora atual, como nome da imagem
271     String name = "Doodlz" + System.currentTimeMillis() + ".jpg";
272
273     // insere a imagem na Galeria do dispositivo
274     String location = MediaStore.Images.Media.insertImage(
275         getContext().getContentResolver(), bitmap, name,
276         "Doodlz Drawing");
277
278     if (location != null) // a imagem foi salva
279     {
280         // exibe uma mensagem indicando que a imagem foi salva
281         Toast message = Toast.makeText(getContext(),
282             R.string.message_saved, Toast.LENGTH_SHORT);
283         message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
284             message.getYOffset() / 2);
285         message.show();
286     }
287     else
288     {
289         // exibe uma mensagem indicando que a imagem não foi salva
290         Toast message = Toast.makeText(getContext(),
291             R.string.message_error_saving, Toast.LENGTH_SHORT);
292         message.setGravity(Gravity.CENTER, message.getXOffset() / 2,

```

Figura 7.32 Método saveImage de DoodleView. (continua)

```
293         message.getYOffset() / 2);
294         message.show();
295     }
296 } // fim do método saveImage
297
```

Figura 7.32 Método `saveImage` de `DoodleView`.

Método `printImage` de `DoodleView`

Nos dispositivos Android 4.4 e mais recentes, o método `printImage` (Fig. 7.33) utiliza a classe `PrintHelper` da Android Support Library para imprimir o desenho atual. A linha 301 primeiramente confirma se, no dispositivo, existe suporte para impressão. Em caso positivo, a linha 304 cria um objeto `PrintHelper`. Em seguida, a linha 307 especifica o *modo de escala* da imagem – `PrintHelper.SCALE_MODE_FIT` indica que a imagem deve caber dentro da área imprimível do papel. Existe também o modo de escala `PrintHelper.SCALE_MODE_FILL`, o qual faz a imagem preencher o papel, possivelmente cortando parte da imagem. Por fim, a linha 308 chama o método `printBitmap` da classe `PrintHelper`, passando como argumentos o nome da tarefa de impressão (usado pela impressora para identificar a impressão) e o objeto `Bitmap` que contém a imagem a ser impressa. Isso exibe a caixa de diálogo de impressão do Android, a qual permite ao usuário escolher se vai salvar a imagem como um documento PDF no dispositivo ou se vai imprimi-la em uma impressora disponível.

```
298 // imprime a imagem atual
299 public void printImage()
300 {
301     if (PrintHelper.systemSupportsPrint())
302     {
303         // usa PrintHelper da Android Support Library para imprimir a imagem
304         PrintHelper printHelper = new PrintHelper(getContext());
305
306         // encaixa a imagem nos limites da página e a imprime
307         printHelper.setScaleMode(PrintHelper.SCALE_MODE_FIT);
308         printHelper.printBitmap("Doodlez Image", bitmap);
309     }
310     else
311     {
312         // exibe uma mensagem indicando que o sistema não permite impressão
313         Toast message = Toast.makeText(getContext(),
314             R.string.message_error_printing, Toast.LENGTH_SHORT);
315         message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
316             message.getYOffset() / 2);
317         message.show();
318     }
319 }
320 } // fim da classe DoodleView
```

Figura 7.33 Método `printImage` de `DoodleView`.

7.7 Classe ColorDialogFragment

A classe `ColorDialogFragment` (Figs. 7.34 a 7.38) estende `DialogFragment` a fim de criar um componente `AlertDialog` para configurar a cor de desenho. As variáveis de instância da classe (linhas 19 a 24) são usadas para referenciar os controles da interface gráfica do usuário para selecionar a nova cor, exibir uma visualização dela e armazená-la como um valor `int` de 32 bits representando os valores ARGB da cor.

```

1 // ColorDialogFragment.java
2 // Permite ao usuário configurar a cor de desenho no elemento DoodleView
3 package com.deitel.doodlz;
4
5 import android.app.Activity;
6 import android.app.AlertDialog;
7 import android.app.Dialog;
8 import android.app.DialogFragment;
9 import android.content.DialogInterface;
10 import android.graphics.Color;
11 import android.os.Bundle;
12 import android.view.View;
13 import android.widget.SeekBar;
14 import android.widget.SeekBar.OnSeekBarChangeListener;
15
16 // classe para a caixa de diálogo Select Color
17 public class ColorDialogFragment extends DialogFragment
18 {
19     private SeekBar alphaSeekBar;
20     private SeekBar redSeekBar;
21     private SeekBar greenSeekBar;
22     private SeekBar blueSeekBar;
23     private View colorView;
24     private int color;
25

```

Figura 7.34 Instrução `package`, instruções `import` e variáveis de instância de `ColorDialogFragment`.

Método sobrescrito `onCreateDialog` de `DialogFragment`

O método `onCreateDialog` (Fig. 7.35) infla o objeto `View` personalizado (linhas 32 a 34) definido por `fragment_color.xml`, que contém a interface gráfica do usuário para selecionar uma cor, e anexa esse objeto `View` ao componente `AlertDialog` chamando o método `setView` de `AlertDialog.Builder` (linha 35). As linhas 42 a 50 obtêm referências para os componentes `SeekBar` e `colorView` da caixa de diálogo. Em seguida, as linhas 53 a 56 registram `colorChangeListener` (Fig. 7.38) como receptor para os eventos dos componentes `SeekBar`.

```

26 // cria um componente AlertDialog e o retorna
27 @Override
28 public Dialog onCreateDialog(Bundle bundle)
29 {
30     AlertDialog.Builder builder =
31         new AlertDialog.Builder(getActivity());
32     View colorDialogView =
33         getActivity().getLayoutInflater().inflate(
34             R.layout.fragment_color, null);

```

Figura 7.35 Método sobrescrito `onCreateDialog` de `DialogFragment`. (continua)

```

35     builder.setView(colorDialogView); // adiciona a interface gráfica do usuário à
                                     // caixa de diálogo
36
37     // configura a mensagem de AlertDialog
38     builder.setTitle(R.string.title_color_dialog);
39     builder.setCancelable(true);
40
41     // obtém os componentes SeekBar de cor e configura seus receptores onChange
42     alphaSeekBar = (SeekBar) colorDialogView.findViewById(
43         R.id.alphaSeekBar);
44     redSeekBar = (SeekBar) colorDialogView.findViewById(
45         R.id.redSeekBar);
46     greenSeekBar = (SeekBar) colorDialogView.findViewById(
47         R.id.greenSeekBar);
48     blueSeekBar = (SeekBar) colorDialogView.findViewById(
49         R.id.blueSeekBar);
50     colorView = colorDialogView.findViewById(R.id.colorView);
51
52     // registra receptores de eventos de SeekBar
53     alphaSeekBar.setOnSeekBarChangeListener(colorChangeListener);
54     redSeekBar.setOnSeekBarChangeListener(colorChangeListener);
55     greenSeekBar.setOnSeekBarChangeListener(colorChangeListener);
56     blueSeekBar.setOnSeekBarChangeListener(colorChangeListener);
57
58     // usa a cor de desenho atual para configurar os valores dos SeekBar
59     final DoodleView doodleView = getDoodleFragment().getDoodleView();
60     color = doodleView.getDrawingColor();
61     alphaSeekBar.setProgress(Color.alpha(color));
62     redSeekBar.setProgress(Color.red(color));
63     greenSeekBar.setProgress(Color.green(color));
64     blueSeekBar.setProgress(Color.blue(color));
65
66     // adiciona o componente Button Set Color
67     builder.setPositiveButton(R.string.button_set_color,
68         new DialogInterface.OnClickListener()
69     {
70         public void onClick(DialogInterface dialog, int id)
71         {
72             doodleView.setDrawingColor(color);
73         }
74     });
75     ); // fim da chamada a setPositiveButton
76
77     return builder.create(); // retorna a caixa de diálogo
78 } // fim do método onCreateDialog
79

```

Figura 7.35 Método sobrescrito onCreateDialog de DialogFragment.

A linha 59 chama o método `getDoodleFragment` (Fig. 7.36) a fim de obter uma referência para o componente `DoodleFragment` e, então, chama o método `getDoodleView` de `DoodleFragment` para obter o elemento `DoodleView`. As linhas 60 a 64 obtêm a cor de desenho atual de `DoodleView` e, então, a utilizam para configurar o valor de cada componente `SeekBar`. Os métodos estáticos `alpha`, `red`, `green` e `blue` de `Color` extraem os valores ARGB da cor, e o método `setProgress` de `SeekBar` posiciona os cursores. As linhas 67 a 75 configuram o botão positivo de `AlertDialog` para configurar a nova cor de desenho de `DoodleView`. A linha 77 retorna o componente `AlertDialog`.

Método `getDoodleFragment`

O método `getDoodleFragment` (Fig. 7.36) simplesmente usa o componente `FragmentManager` a fim de obter uma referência para `DoodleFragment`.

```

80    // obtém uma referência para o componente DoodleFragment
81    private DoodleFragment getDoodleFragment()
82    {
83        return (DoodleFragment) getFragmentManager().findFragmentById(
84            R.id.doodleFragment);
85    }
86

```

Figura 7.36 Método `getDoodleFragment`.

Métodos sobrescritos de ciclo de vida `onAttach` e `onDetach` de `Fragment`

Quando o componente `ColorDialogFragment` é adicionado a uma atividade pai, o método `onAttach` (Fig. 7.37, linhas 88 a 96) é chamado. A linha 92 obtém uma referência para o componente `DoodleFragment`. Se essa referência não é nula, a linha 95 chama o método `setDialogOnScreen` de `DoodleFragment` para indicar que a caixa de diálogo **Choose Color** está sendo exibida no momento. Quando o componente `ColorDialogFragment` é removido de uma atividade pai, o método `onDetach` (linhas 99 a 107) é chamado. A linha 106 chama o método `setDialogOnScreen` de `DoodleFragment` para indicar que a caixa de diálogo **Choose Color** não está mais na tela.

```

87    // informa DoodleFragment de que a caixa de diálogo está sendo exibida
88    @Override
89    public void onAttach(Activity activity)
90    {
91        super.onAttach(activity);
92        DoodleFragment fragment = getDoodleFragment();
93
94        if (fragment != null)
95            fragment.setDialogOnScreen(true);
96    }
97
98    // informa DoodleFragment de que a caixa de diálogo não está mais sendo exibida
99    @Override
100    public void onDetach()
101    {
102        super.onDetach();
103        DoodleFragment fragment = getDoodleFragment();
104
105        if (fragment != null)
106            fragment.setDialogOnScreen(false);
107    }
108

```

Figura 7.37 Métodos sobrescritos de ciclo de vida `onAttach` e `onDetach` de `Fragment`.

Classe interna anônima que implementa a interface `OnSeekBarChangeListener` para responder aos eventos dos componentes `SeekBar` para alfa, vermelho, verde e azul

A Figura 7.38 define uma classe interna anônima que implementa a interface `OnSeekBarChangeListener` para responder aos eventos quando o usuário ajusta os componentes `SeekBar` da caixa de diálogo **Dialog Choose Color**. Isso foi registrado como rotina de tratamento

de eventos dos componentes SeekBar na Figura 7.35 (linhas 53 a 56). O método `onProgressChanged` (linhas 115 a 123) é chamado quando a posição do cursor de um componente SeekBar muda. Se o usuário moveu o cursor de um componente SeekBar (linha 118), as linhas 119 a 121 armazenam a nova cor. O método estático `argb` da classe `Color` combina os valores dos componentes SeekBar em um objeto `Color` e retorna a cor apropriada como um valor `int`. Então, usamos o método `setBackgroundColor` da classe `View` para atualizar o elemento `colorView` com a cor correspondente ao estado dos componentes SeekBar.

```

109 // OnSeekBarChangeListener para os componentes SeekBar na caixa de diálogo de cor
110 private OnSeekBarChangeListener colorChangeListener =
111     new OnSeekBarChangeListener()
112     {
113         // exibe a cor atualizada
114         @Override
115         public void onProgressChanged(SeekBar seekBar, int progress,
116             boolean fromUser)
117         {
118             if (fromUser) // o usuário, não o programa, alterou o cursor do
119                 // componente SeekBar
120                 color = Color.argb(alphaSeekBar.getProgress(),
121                     redSeekBar.getProgress(), greenSeekBar.getProgress(),
122                     blueSeekBar.getProgress());
123             colorView.setBackgroundColor(color);
124         }
125         @Override
126         public void onStartTrackingTouch(SeekBar seekBar) // obrigatório
127         {
128         }
129         @Override
130         public void onStopTrackingTouch(SeekBar seekBar) // obrigatório
131         {
132         }
133     }; // fim de colorChanged
134 } // fim da classe ColorDialogFragment

```

Figura 7.38 Classe interna anônima que implementa a interface `OnSeekBarChangeListener` para responder aos eventos dos componentes SeekBar para alfa, vermelho, verde e azul.

7.8 Classe `LineWidthDialogFragment`

A classe `LineWidthDialogFragment` (Fig. 7.39) estende `DialogFragment` a fim de criar um componente `AlertDialog` para configurar a largura da linha. Ela é semelhante à classe `ColorDialogFragment`; portanto, discutimos aqui somente as principais diferenças. A única variável de instância da classe é um elemento `ImageView` (linha 22), no qual desenhamos uma linha mostrando a configuração de largura de linha atual.

```

1 // LineWidthDialogFragment.java
2 // Permite ao usuário configurar a cor de desenho no elemento DoodleView
3 package com.deitel.doodlz;
4
5 import android.app.Activity;
6 import android.app.AlertDialog;

```

Figura 7.39 Classe `LineWidthDialogFragment`.

```

7 import android.app.Dialog;
8 import android.app.DialogFragment;
9 import android.content.DialogInterface;
10 import android.graphics.Bitmap;
11 import android.graphics.Canvas;
12 import android.graphics.Paint;
13 import android.os.Bundle;
14 import android.view.View;
15 import android.widget.ImageView;
16 import android.widget.SeekBar;
17 import android.widget.SeekBar.OnSeekBarChangeListener;
18
19 // classe para a caixa de diálogo Select Color
20 public class LineWidthDialogFragment extends DialogFragment
21 {
22     private ImageView widthImageView;
23
24     // cria um componente AlertDialog e o retorna
25     @Override
26     public Dialog onCreateDialog(Bundle bundle)
27     {
28         AlertDialog.Builder builder =
29             new AlertDialog.Builder(getActivity());
30         View lineWidthDialogView = getActivity().getLayoutInflater().inflate(
31             R.layout.fragment_line_width, null);
32         builder.setView(lineWidthDialogView); // adiciona a interface gráfica do
                                                // usuário à caixa de diálogo
33
34         // configura a mensagem de AlertDialog
35         builder.setTitle(R.string.title_line_width_dialog);
36         builder.setCancelable(true);
37
38         // obtém o componente ImageView
39         widthImageView = (ImageView) lineWidthDialogView.findViewById(
40             R.id.widthImageView);
41
42         // configura widthSeekBar
43         final DoodleView doodleView = getDoodleFragment().getDoodleView();
44         final SeekBar widthSeekBar = (SeekBar)
45             lineWidthDialogView.findViewById(R.id.widthSeekBar);
46         widthSeekBar.setOnSeekBarChangeListener(lineWidthChanged);
47         widthSeekBar.setProgress(doodleView.getLineWidth());
48
49         // adiciona o componente Button Set Line Width
50         builder.setPositiveButton(R.string.button_set_line_width,
51             new DialogInterface.OnClickListener()
52             {
53                 public void onClick(DialogInterface dialog, int id)
54                 {
55                     doodleView.setLineWidth(widthSeekBar.getProgress());
56                 }
57             }
58         ); // fim da chamada a setPositiveButton
59
60         return builder.create(); // retorna a caixa de diálogo
61     } // fim do método onCreateDialog
62
63     // obtém uma referência para o componente DoodleFragment
64     private DoodleFragment getDoodleFragment()
65     {
66         return (DoodleFragment) getFragmentManager().findFragmentById(

```

Figura 7.39 Classe LineWidthDialogFragment. (continua)

```

67         R.id.doodleFragment);
68     }
69
70     // informa DoodleFragment de que a caixa de diálogo está sendo exibida
71     @Override
72     public void onAttach(Activity activity)
73     {
74         super.onAttach(activity);
75         DoodleFragment fragment = getDoodleFragment();
76
77         if (fragment != null)
78             fragment.setDialogOnScreen(true);
79     }
80
81     // informa DoodleFragment de que a caixa de diálogo não está mais sendo exibida
82     @Override
83     public void onDetach()
84     {
85         super.onDetach();
86         DoodleFragment fragment = getDoodleFragment();
87
88         if (fragment != null)
89             fragment.setDialogOnScreen(false);
90     }
91
92     // OnSeekBarChangeListener para o componente SeekBar na caixa de diálogo de largura
93     private OnSeekBarChangeListener lineWidthChanged =
94         new OnSeekBarChangeListener()
95         {
96             Bitmap bitmap = Bitmap.createBitmap(
97                 400, 100, Bitmap.Config.ARGB_8888);
98             Canvas canvas = new Canvas(bitmap); // associa ao Canvas
99
100             @Override
101             public void onProgressChanged(SeekBar seekBar, int progress,
102                 boolean fromUser)
103             {
104                 // configura um objeto Paint para o valor de SeekBar atual
105                 Paint p = new Paint();
106                 p.setColor(
107                     getDoodleFragment().getDoodleView().getDrawingColor());
108                 p.setStrokeCap(Paint.Cap.ROUND);
109                 p.setStrokeWidth(progress);
110
111                 // apaga o bitmap e redesenha a linha
112                 bitmap.eraseColor(
113                     getResources().getColor(android.R.color.transparent));
114                 canvas.drawLine(30, 50, 370, 50, p);
115                 widthImageView.setImageBitmap(bitmap);
116             }
117
118             @Override
119             public void onStartTrackingTouch(SeekBar seekBar) // obrigatório
120             {
121             }
122
123             @Override
124             public void onStopTrackingTouch(SeekBar seekBar) // obrigatório
125             {
126             }
127         }; // fim de lineWidthChanged
128 }

```

Figura 7.39 Classe LineWidthDialogFragment.

Método onCreateDialog

O método `onCreateDialog` (linhas 25 a 61) infla o objeto `View` personalizado (linhas 30 e 31) definido por `fragment_line_width.xml`, o qual exibe a interface gráfica do usuário para selecionar a largura da linha, e anexa esse objeto `View` ao componente `AlertDialog` chamando o método `setView` de `AlertDialog.Builder` (linha 32). As linhas 39 e 40 obtêm uma referência para o elemento `ImageView` no qual a amostra da linha será desenhada. Em seguida, as linhas 43 a 47 obtêm uma referência para o elemento `widthSeekBar`, registram `lineWidthChanged` (linhas 93 a 127) como receptor de `SeekBar` e configuram o valor atual do componente `SeekBar` com a largura de linha atual. As linhas 50 a 58 definem o botão positivo da caixa de diálogo para chamar o método `setLineWidth` de `DoodleView` quando o usuário tocar no botão **Set Line Width**. A linha 60 retorna o componente `AlertDialog` para exibição.

Classe interna anônima que implementa a interface

OnSeekBarChangeListener para responder aos eventos do componente widthSeekBar

As linhas 93 a 127 definem o receptor `OnSeekBarChangeListener` de `lineWidthChanged` que responde aos eventos quando o usuário ajusta o componente `SeekBar` na caixa de diálogo **Choose Line Width**. As linhas 96 e 97 criam um objeto `Bitmap` para exibir uma amostra da linha representando a espessura selecionada. A linha 98 cria um `Canvas` para desenhar no objeto `Bitmap`. O método `onProgressChanged` (linhas 100 a 116) desenha a amostra de linha com base na cor de desenho atual e no valor do componente `SeekBar`. Primeiramente, as linhas 105 a 109 configuram um objeto `Paint` para desenhar a amostra de linha. O método `setStrokeCap` da classe `Paint` (linha 108) especifica a aparência das extremidades da linha – neste caso, elas são arredondadas (`Paint.Cap.ROUND`). As linhas 112 e 113 limpam o fundo do `bitmap` com a cor `android.R.color.transparent` predefinida do Android com o método `eraseColor` de `Bitmap`. Usamos `canvas` para desenhar a amostra de linha. Por fim, a linha 115 exibe o `bitmap` em `widthImageView`, passando-o para o método `setImageBitmap` de `ImageView`.

7.9 Classe EraseImageDialogFragment

A classe `EraseImageDialogFragment` (Fig. 7.40) estende `DialogFragment` para criar um componente `AlertDialog` que confirma se o usuário quer realmente apagar a imagem inteira. Ela é semelhante às classes `ColorDialogFragment` e `LineWidthDialogFragment`; portanto, discutimos aqui somente o método `onCreateDialog` (linhas 16 a 41). O método cria um componente `AlertDialog` com botões **Erase Image** e **Cancel**. As linhas 27 a 35 configuram **Erase Image** como o botão positivo – quando o usuário toca nele, a linha 32 no receptor do botão chama o método `clear` de `DoodleView` para apagar a imagem. A linha 38 configura **Cancel** como o botão negativo – quando o usuário toca nele, a caixa de diálogo é descartada. A linha 40 retorna o componente `AlertDialog`.

```
1 // EraseImageDialogFragment.java
2 // Permite ao usuário apagar a imagem
3 package com.deitel.doodlz;
4
5 import android.app.Activity;
6 import android.app.AlertDialog;
```

Figura 7.40 Classe `EraseImageDialogFragment`. (continua)

```
7  import android.app.Dialog;
8  import android.app.DialogFragment;
9  import android.content.DialogInterface;
10 import android.os.Bundle;
11
12 // classe para a caixa de diálogo Select Color
13 public class EraseImageDialogFragment extends DialogFragment
14 {
15     // cria um componente AlertDialog e o retorna
16     @Override
17     public Dialog onCreateDialog(Bundle bundle)
18     {
19         AlertDialog.Builder builder =
20             new AlertDialog.Builder(getActivity());
21
22         // configura a mensagem de AlertDialog
23         builder.setMessage(R.string.message_erase);
24         builder.setCancelable(false);
25
26         // adiciona o componente Button Erase
27         builder.setPositiveButton(R.string.button_erase,
28             new DialogInterface.OnClickListener()
29             {
30                 public void onClick(DialogInterface dialog, int id)
31                 {
32                     getDoodleFragment().getDoodleView().clear(); // apaga a imagem
33                 }
34             }); // fim da chamada a setPositiveButton
35
36         // adiciona o componente Button Cancel
37         builder.setNegativeButton(R.string.button_cancel, null);
38
39         return builder.create(); // retorna a caixa de diálogo
40     } // fim do método onCreateDialog
41
42     // obtém uma referência para o componente DoodleFragment
43     private DoodleFragment getDoodleFragment()
44     {
45         return (DoodleFragment) getFragmentManager().findFragmentById(
46             R.id.doodleFragment);
47     }
48
49     // informa DoodleFragment de que a caixa de diálogo está sendo exibida
50     @Override
51     public void onAttach(Activity activity)
52     {
53         super.onAttach(activity);
54         DoodleFragment fragment = getDoodleFragment();
55
56         if (fragment != null)
57             fragment.setDialogOnScreen(true);
58     }
59
60     // informa DoodleFragment de que a caixa de diálogo não está mais sendo exibida
61     @Override
62     public void onDetach()
63     {
64         super.onDetach();
65         DoodleFragment fragment = getDoodleFragment();
66     }
67 }
```

Figura 7.40 Classe EraseImageDialogFragment. (continua)

```

68         if (fragment != null)
69             fragment.setDialogOnScreen(false);
70     }
71 } // fim da classe EraseImageDialogFragment

```

Figura 7.40 Classe EraseImageDialogFragment.

7.10 Para finalizar

Neste capítulo, você construiu o aplicativo **Doodlz**, que permite aos usuários pintar arrastando um ou mais dedos pela tela. Você implementou um recurso de chacoalhar para apagar usando o componente **SensorManager** do Android para registrar um receptor **SensorEventListener** que responde a eventos de acelerômetro, e aprendeu que o Android suporta muitos outros sensores.

Você criou subclasses de **DialogFragment** que exibiam objetos **View** personalizados em componentes **AlertDialog**. Também sobrescreveu os métodos de ciclo de vida **onAttach** e **onDetach** de **Fragment**, os quais são chamados quando um fragmento é anexado ou desanexado de uma atividade pai, respectivamente.

Mostramos como associar um **Canvas** a um **Bitmap** e, então, usar o **Canvas** para desenhar no **Bitmap**. Demonstramos como tratar eventos multitouch para que o usuário possa desenhar simultaneamente com vários dedos. Você armazenou as informações de cada dedo como um objeto **Path**. Processou os eventos de toque sobrescrevendo o método **onTouchEvent** de **View**, o qual recebe um parâmetro **MotionEvent** contendo o tipo de evento e o identificador do ponteiro que o gerou. Usamos os identificadores para distinguir os diferentes dedos e adicionamos as informações nos objetos **Path** correspondentes.

Você usou o novo modo imersivo do Android 4.4, que possibilita a um aplicativo utilizar a tela inteira, mas ainda permite ao usuário acessar as barras do sistema, quando necessário. Para alternar para o modo imersivo, você usou um elemento **GestureDetector** a fim de determinar quando o usuário deu um toque rápido na tela.

Você usou um elemento **ContentResolver** e o método **MediaStore.Images.Media.insertImage** para salvar uma imagem na **Galeria** do dispositivo. Por último, mostramos como usar o novo framework de impressão do Android 4.4 para permitir aos usuários imprimir seus desenhos. Você usou a classe **PrintHelper** da Android Support Library para imprimir um **bitmap**. A classe **PrintHelper** exibiu uma interface de usuário para selecionar uma impressora ou salvar a imagem em um documento PDF.

No Capítulo 8, construiremos o aplicativo **Address Book**, voltado a bancos de dados, o qual oferece acesso rápido e fácil às informações de contato armazenadas e capacidade de adicionar, excluir e editar contatos. Você vai aprender a alternar entre fragmentos dinamicamente em uma interface gráfica de usuário e, mais uma vez, a fornecer layouts que otimizam o espaço disponível na tela em telefones e tablets.

Exercícios de revisão

7.1 Preencha os espaços em branco em cada um dos seguintes enunciados:

- Você usa o componente **SensorManager** para registrar as mudanças no sensor que seu aplicativo deve receber e para especificar o _____ que irá tratar esses eventos de alteração no sensor.
- Um objeto **Path** (pacote **android.graphics**) representa um caminho geométrico consistindo em segmentos de linha e _____.

- c) Você usa o tipo do evento de toque para determinar se o usuário tocou na tela, _____ ou tirou o dedo da tela.
- d) Use o método _____ da classe `SensorManager` para parar de detectar eventos de acelerômetro.
- e) Sobrescreva o método _____ de `SensorEventListener` para processar eventos de acelerômetro.
- f) Sobrescreva o método _____ de `Fragment` para responder ao evento quando um fragmento é anexado a uma atividade pai.
- g) Quando uma `View` precisa ser redesenhada, seu método _____ é chamado.
- h) O método _____ de `MotionEvent` retorna um valor `int` representando o tipo de `MotionEvent`, o qual você pode usar com constantes da classe `MotionEvent` para determinar como vai tratar cada evento.
- i) O _____ do Android 4.4 permite a um aplicativo aproveitar a tela inteira.

7.2 Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.

- a) Você anula o registro da rotina de tratamento de evento de acelerômetro quando o aplicativo é enviado para o primeiro plano.
- b) Chame o método herdado `validate` de `View` para indicar que o elemento `View` precisa ser redesenhado.
- c) Se a ação for `MotionEvent.ACTION_DOWN` ou `MotionEvent.ACTION_POINTER_DOWN`, o usuário tocou na tela com o mesmo dedo.
- d) Redefinir `Path` apaga sua linha pintada correspondente da tela, pois essas linhas já foram desenhadas no `bitmap` que está sendo exibido.
- e) O método `MediaStore.Images.Media.saveImage` salva um `Bitmap` na galeria (**Gallery**) do dispositivo.

Respostas dos exercícios de revisão

- 7.1** a) `SensorEventListener`. b) curvas. c) arrastou o dedo pela tela. d) `unregisterListener`. e) `onSensorChanged`. f) `onAttach`. g) `onDraw`. h) `getActionMasked`. i) modo imersivo.
- 7.2** a) Falsa. Você anula o registro da rotina de tratamento de evento de acelerômetro quando o aplicativo é enviado para o *segundo plano*. b) Falsa. Chame o método herdado `invalidate` de `View` para indicar que o elemento `View` precisa ser redesenhado. c) Falsa. Se a ação for `MotionEvent.ACTION_DOWN` ou `MotionEvent.ACTION_POINTER_DOWN`, o usuário tocou na tela com um novo dedo. d) Falsa. Redefinir `Path` *não apaga* sua linha pintada correspondente da tela, pois essas linhas já foram desenhadas no `bitmap` que está sendo exibido. e) Falsa. O método `MediaStore.Images.Media.insertImage` salva um `Bitmap` na galeria (**Gallery**) do dispositivo.

Exercícios

- 7.3** Preencha os espaços em branco em cada um dos seguintes enunciados:
 - a) A maioria dos dispositivos Android tem um _____ que permite aos aplicativos detectar movimentos.
 - b) Sobrescreva o método _____ de `Fragment` para responder ao evento quando um fragmento é anexado a uma atividade pai.
 - c) O _____ monitora o acelerômetro para detectar movimentos no dispositivo.
 - d) A constante _____ de `SensorManager` representa a aceleração em virtude da gravidade da Terra.
 - e) Você se registra para receber eventos de acelerômetro usando o método `registerListener` de `SensorManager`, o qual recebe três argumentos: o objeto `SensorEventListener` que vai

responder aos eventos, um `Sensor` representando o tipo de dados de sensor que o aplicativo deseja receber e _____.

- f) Você passa `true` para o método _____ de `Paint` para habilitar o anti-aliasing que suaviza as bordas das linhas.
- g) O método _____ de `Paint` configura a largura do traço com o número especificado de pixels.
- h) O Android suporta _____ – isto é, vários dedos tocando na tela.
- i) A classe _____ da Android Support Library fornece uma interface gráfica do usuário para selecionar uma impressora e o método _____ para imprimir um `Bitmap`.

7.4 Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.

- a) No Android, os eventos de sensor são tratados na thread da interface gráfica do usuário.
- b) O componente `alpha` especifica a transparência do elemento `Color`, com 0 representando completamente transparente e 100 representando completamente opaco.
- c) Para eventos de acelerômetro, o array `values` do parâmetro `SensorEvent` contém três elementos representando a aceleração (em metros/segundo²) nas direções *x* (esquerda/direita), *y* (para cima/para baixo) e *z* (para frente/para trás).
- d) O método `onProgressChanged` é chamado uma vez quando o usuário arrasta o cursor de um componente `SeekBar`.
- e) Para obtermos o identificador exclusivo do dedo, que persiste entre os objetos `MotionEvent` até que o usuário retire esse dedo da tela, você usa o método `getID` de `MotionEvent`, passando o índice do dedo como argumento.
- f) O evento de sistema `MotionEvent` passado a partir de `onTouchEvent` contém informações de toque para vários movimentos na tela, caso ocorram ao mesmo tempo.

7.5 (*Aplicativo Doodlz melhorado*) Faça as seguintes melhorias no aplicativo `Doodlz`:

- a) Permita que o usuário selecione uma cor de fundo. O recurso de apagamento deve usar a cor de fundo selecionada. O apagamento da imagem inteira deve retornar o fundo para a cor branca padrão.
- b) Permita que o usuário selecione uma imagem de fundo para desenhar sobre ela. O apagamento da imagem inteira deve retornar o fundo para a cor branca padrão. O recurso de apagamento deve usar a cor de fundo branca padrão.
- c) Use pressão para determinar a transparência da cor ou da espessura da linha. A classe `MotionEvent` tem métodos que permitem obter a pressão do toque.
- d) Acrescente a capacidade de desenhar retângulos e elipses. Deve haver opções para preencher a forma ou deixá-la vazada. O usuário deve ter a capacidade de especificar a espessura da linha da borda e a cor de preenchimento de cada forma.
- e) (*Avançado*) Quando o usuário selecionar uma imagem de fundo sobre a qual quer desenhar, o recurso de apagamento deve revelar os pixels da imagem de fundo original no local apagado.

7.6 (*Aplicativo Hangman Game*) Recrie o clássico `Hangman Game` (Jogo da Forca) usando o ícone de robô do Android em vez de um bonequinho. (Para ver os termos de uso do logotipo do Android, visite www.android.com/branding.html). No início do jogo, exiba uma linha tracejada, com um traço representando cada letra da palavra. Como dica para o usuário, forneça a categoria (por exemplo, esporte ou ponto de referência) ou a definição da palavra. Peça ao jogador para digitar uma letra. Se a letra existe na palavra, coloque-a no lugar do traço correspondente. Se ela não faz parte da palavra, desenhe parte do robô do Android na tela (por exemplo, a cabeça do robô). Para cada resposta incorreta, desenhe outra parte do robô do Android. O jogo termina quando o usuário completa a palavra ou o robô do Android inteiro é desenhado na tela.

- 7.7 (*Aplicativo Fortune Teller*)** O usuário “faz uma pergunta” e então chacoalha o telefone para saber qual é o seu futuro (por exemplo, “provavelmente não”, “parece promissor”, “pergunte-me depois” etc).
- 7.8 (*Jogo Block Breaker*)** Exiba várias colunas de blocos nas cores vermelha, amarela, azul e verde. Cada coluna deve ter blocos de cada cor colocados aleatoriamente. Os blocos só podem ser removidos da tela se estiverem em grupos de dois ou mais. Um grupo consiste em blocos adjacentes da mesma cor dispostos verticalmente e/ou horizontalmente. Quando o usuário toca em um grupo de blocos, o grupo desaparece e os blocos de cima se movem para baixo a fim de preencher o espaço. O objetivo é eliminar todos os blocos da tela. Mais pontos devem ser dados para grupos de blocos maiores.
- 7.9 (*Aplicativo Block Breaker melhorado*)** Modifique o aplicativo **Block Breaker** do Exercício 7.8 como segue:
- a) Forneça um cronômetro — o usuário vence eliminando os blocos no tempo designado. Aumente o número de blocos na tela se o usuário levar mais tempo para limpá-la.
 - b) Adicione vários níveis. Em cada nível, o tempo designado para limpar a tela diminui.
 - c) Forneça um modo contínuo no qual, à medida que o usuário eliminar blocos, uma nova fileira de blocos seja acrescentada. Se o espaço abaixo de determinado bloco estiver vazio, o bloco desce para preencher esse espaço. Nesse modo, o jogo termina quando o usuário não puder remover mais blocos.
 - d) Monitore as pontuações mais altas em cada modo do jogo.
- 7.10 (*Aplicativo Word Search*)** Crie uma grade de letras que preencha a tela. Na grade, devem estar ocultas pelo menos 10 palavras. As palavras podem estar na horizontal, na vertical ou na diagonal e, em cada caso, para frente, para trás, para cima ou para baixo. Permita ao usuário destacar as palavras arrastando o dedo pelas letras na tela ou tocando em cada letra da palavra. Inclua um cronômetro. Quanto menos tempo o usuário demorar para terminar o jogo, maior a pontuação. Registre as pontuações mais altas.
- 7.11 (*Aplicativo Fractal*)** Pesquise como se faz para desenhar fractais e desenvolva um aplicativo que os desenhe. Forneça opções que permitam ao usuário controlar o número de níveis do fractal e suas cores.
- 7.12 (*Aplicativo Kaleidoscope*)** Crie um aplicativo que simule um caleidoscópio. Permita que o usuário chacoalhe o dispositivo para redesenhar a tela.
- 7.13 (*Aplicativo Labyrinth Game: código-fonte aberto*)** Examine o aplicativo Android de código-fonte aberto *Amazed* no site Google Code (<http://apps-for-android.id.googlesource.com/svn/trunk/Amazed/>). Nesse jogo, o usuário manobra uma bolinha por um labirinto, inclinando o dispositivo em várias direções. As modificações e melhorias possíveis incluem: adicionar um cronômetro para monitorar a rapidez com que o usuário termina o jogo, melhorar os elementos gráficos, adicionar sons e adicionar mais quebra-cabeças de variados graus de dificuldade.
- 7.14 (*Aplicativo Game of Snake*)** Pesquise o Game of Snake online e desenvolva um aplicativo que permita ao usuário jogá-lo.



Objetivos

Neste capítulo, você vai:

- Usar um componente `ListFragment` para exibir e gerenciar um elemento `ListView`.
- Usar componentes `FragmentTransaction` e a pilha de retrocesso para anexar e desanexar dinamicamente fragmentos da interface gráfica do usuário.
- Criar e abrir bancos de dados SQLite usando um elemento `SQLiteOpenHelper` e inserir, excluir e consultar dados em um banco de dados SQLite usando um objeto `SQLiteDatabase`.
- Usar um objeto `SimpleCursorAdapter` para vincular resultados de consulta de banco de dados aos itens de um elemento `ListView`.
- Usar um objeto `Cursor` para manipular resultados de uma consulta de banco de dados.
- Usar múltiplas threads e elementos `AsyncTask` para efetuar operações de banco de dados fora da thread da interface gráfica do usuário e manter a rapidez de resposta do aplicativo.
- Definir estilos contendo atributos e valores comuns de interface gráfica do usuário e, em seguida, aplicá-los em vários componentes da interface.

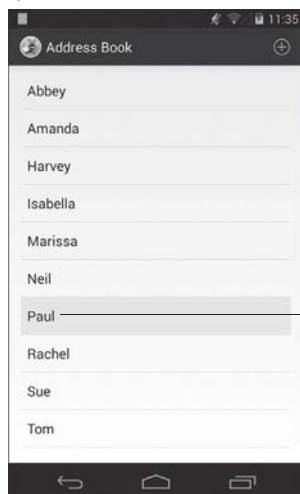
- 8.1 Introdução
- 8.2 Teste do aplicativo **Address Book**
- 8.3 Visão geral das tecnologias
 - 8.3.1 Exibição de fragmentos com componentes `FragmentManager`
 - 8.3.2 Comunicação de dados entre um fragmento e uma atividade hospedeira
 - 8.3.3 Método `onSaveInstanceState`
 - 8.3.4 Definindo estilos e aplicando-os nos componentes da interface gráfica do usuário
 - 8.3.5 Especificação de um fundo para um componente `TextView`
 - 8.3.6 Extensão da classe `ListFragment` para criar um fragmento contendo um componente `ListView`
 - 8.3.7 Manipulação de um banco de dados `SQLite`
 - 8.3.8 Execução de operações de banco de dados fora da thread da interface gráfica do usuário com elementos `AsyncTask`
- 8.4 Construção da interface gráfica do usuário e do arquivo de recursos
 - 8.4.1 Criação do projeto
 - 8.4.2 Criação das classes do aplicativo
 - 8.4.3 `strings.xml`
 - 8.4.4 `styles.xml`
 - 8.4.5 `textview_border.xml`
 - 8.4.6 Layout de `MainActivity`: `activity_main.xml`
 - 8.4.7 Layout de `DetailsFragment`: `fragment_details.xml`
 - 8.4.8 Layout de `AddEditFragment`: `fragment_add_edit.xml`
 - 8.4.9 Definição dos menus dos fragmentos
- 8.5 Classe `MainActivity`
- 8.6 Classe `ContactListFragment`
- 8.7 Classe `AddEditFragment`
- 8.8 Classe `DetailsFragment`
- 8.9 Classe utilitária `DatabaseConnector`
- 8.10 Para finalizar

Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios

8.1 Introdução

O aplicativo **Address Book** (Fig. 8.1) fornece acesso a informações de contato armazenadas em um banco de dados `SQLite` no dispositivo. Você pode rolar por uma lista de contatos em ordem alfabética e ver os detalhes de cada contato tocando em seu nome.

a) Lista de contatos com **Paul** selecionado



Tocar no nome de um contato exibe um fragmento contendo suas informações detalhadas.

b) Detalhes do contato **Paul**

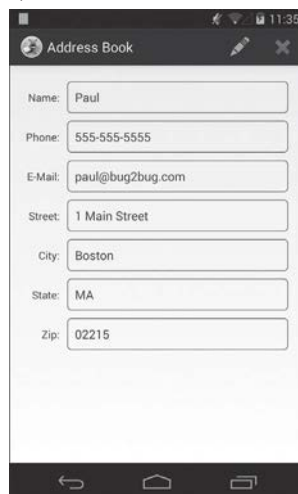
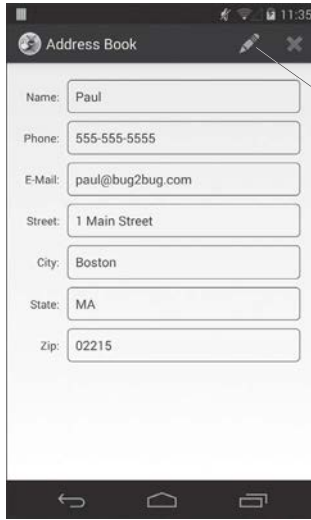


Figura 8.1 Lista de contatos e os detalhes de um contato selecionado.

Quando os detalhes de um contato são exibidos, tocar em *editar* (✎) exibe um fragmento contendo componentes `EditText` previamente preenchidos, para edição dos dados do contato (Fig. 8.2). Tocar em *excluir* (✕) exibe um componente `DialogFragment` pedindo ao usuário para que confirme a exclusão (Fig. 8.3).

a) Toque no ícone de edição para editar o contato atual



Tocar no ícone de edição na barra de ação exibe um fragmento para editar os dados desse contato

b) Fragmento para editar o contato

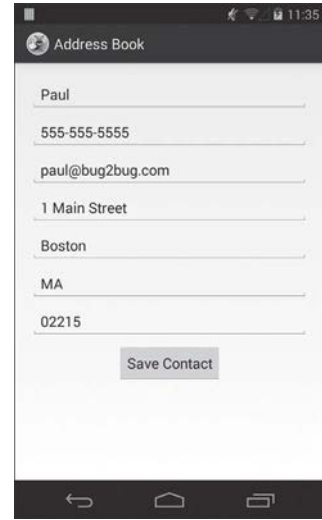


Figura 8.2 Editando os dados de um contato.

a) Toque no ícone de exclusão para excluir o contato atual



Tocar no ícone de exclusão na barra de ação exibe uma caixa de diálogo pedindo ao usuário para que confirme a exclusão

b) Caixa de diálogo de confirmação para excluir o contato

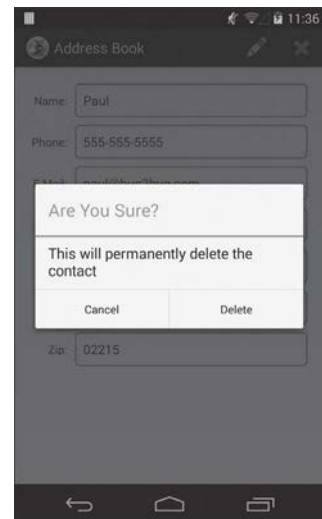
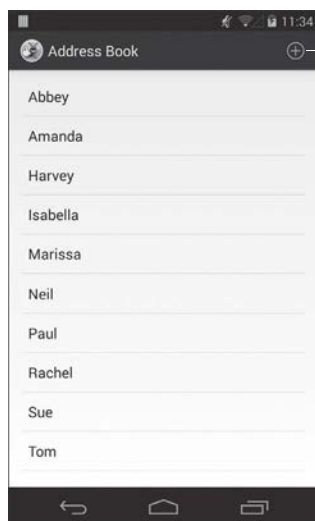


Figura 8.3 Excluindo um contato do banco de dados.

Ao visualizar a lista de contatos, se tocar em *adicionar* (+) será exibido um fragmento contendo componentes `EditText` que podem ser usados para adicionar os dados do novo contato (Fig. 8.4). Ao editar um contato já existente ou adicionar um novo, você toca no botão **Save Contact** para salvar os dados do contato. A Figura 8.5 mostra o aplicativo sendo executado em um tablet na orientação paisagem. Em tablets, a lista de contatos é sempre exibida no lado esquerdo do aplicativo.

a) Toque no ícone de adição para adicionar um novo contato



Tocar no ícone de adição na barra de ação exibe um fragmento para inserir um novo contato

b) Fragmento para adicionar o contato

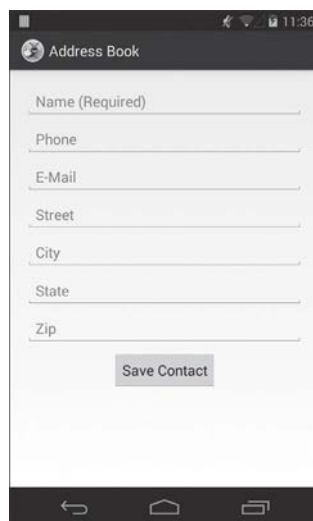


Figura 8.4 Adicionando um contato ao banco de dados.

a) Na orientação paisagem, em um telefone ou tablet, os ícones da barra de ação aparecem com seus textos

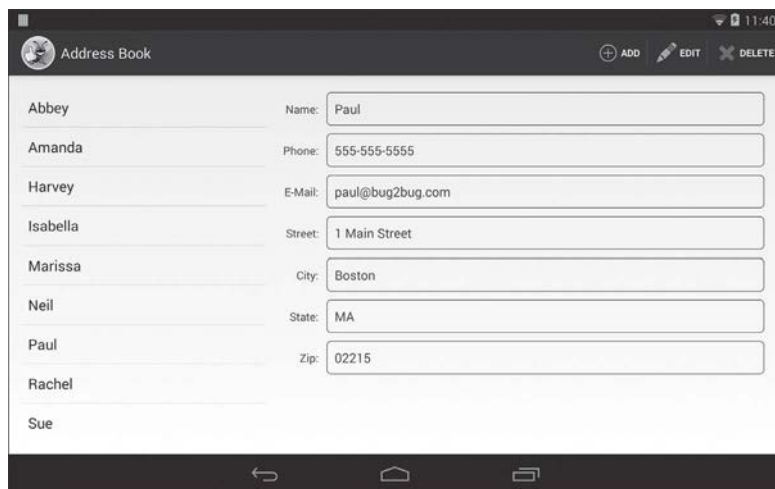


Figura 8.5 O aplicativo **Address Book** sendo executado no modo paisagem em um tablet.

8.2 Teste do aplicativo Address Book

Abriendo e executando o aplicativo

Abra o Eclipse e importe o projeto do aplicativo **Address Book**. Execute os passos a seguir:

1. **Abra a caixa de diálogo Import.** Selecione **File > Import...** para abrir a caixa de diálogo **Import**.
2. **Importe o projeto do aplicativo Address Book.** Na caixa de diálogo **Import**, expanda o nó **General** e selecione **Existing Projects into Workspace**; em seguida, clique em **Next >** para passar à etapa **Import Projects**. Certifique-se de que **Select root directory** esteja selecionado e, em seguida, clique no botão **Browse....** Na caixa de diálogo **Browse for Folder**, localize a pasta **AddressBook** na pasta de exemplos do livro, selecione-a e clique em **OK**. Clique em **Finish** a fim de importar o projeto para o Eclipse. Agora o projeto aparece na janela **Package Explorer**, no lado esquerdo da janela do Eclipse.
3. **Ative o aplicativo Address Book.** No Eclipse, clique com o botão direito do mouse no projeto **AddressBook** na janela **Package Explorer** e, em seguida, selecione **Run As > Android Application** no menu que aparece.

Adicionando um contato

Na primeira vez que você executar o aplicativo, a lista de contatos estará vazia e exibirá **No Contacts** no centro da tela. Toque em  na barra de ação a fim de exibir a tela para adicionar uma nova entrada. Após adicionar as informações do contato, toque no botão **Save Contact** para armazenar o contato no banco de dados e voltar à tela principal do aplicativo. Se optar por não adicionar o contato, você pode simplesmente tocar no botão *voltar* do dispositivo para voltar à tela principal. Adicione mais contatos, se desejar. Em um tablet, após a adição de um contato, os detalhes do novo contato aparecerão à direita da lista de contatos, como na Fig. 8.5.

Visualizando um contato

Para ver os detalhes de um contato, toque no nome do contato que você acabou de adicionar na lista de contatos. Em um tablet, os detalhes aparecem à direita da lista de contatos.

Editando um contato

Enquanto estiver vendo os detalhes de um contato, toque em  na barra de ação para exibir uma tela de componentes **EditText** previamente preenchidos com dados de contato. Edite os dados conforme for necessário e, em seguida, toque no botão **Save Contact** para armazenar as informações de contato atualizadas no banco de dados e voltar à tela principal do aplicativo. Em um tablet, após a edição de um contato, os detalhes do novo contato aparecerão à direita da lista de contatos.

Excluindo um contato

Enquanto estiver vendo os detalhes de um contato, toque em  na barra de ação para excluir o contato. Vai aparecer uma caixa de diálogo pedindo que você confirme essa ação. Se você confirmar, o contato será removido do banco de dados e o aplicativo exibirá a lista de contatos atualizada.

8.3 Visão geral das tecnologias

Esta seção apresenta as novas tecnologias que usamos no aplicativo **Address Book**, na ordem em que são encontradas ao longo do capítulo.

8.3.1 Exibição de fragmentos com componentes **FragmentManager**

Em aplicativos anteriores que usavam fragmentos, você declarou cada fragmento no layout de uma atividade ou, para um componente **DialogFragment**, chamou seu método `show` para criá-lo. O aplicativo **Flag Quiz** demonstrou como usar várias atividades para armazenar cada um dos fragmentos do aplicativo em um telefone. Neste aplicativo, você vai usar apenas uma atividade para armazenar todos os fragmentos. Em um dispositivo do tamanho de um telefone, você vai exibir um fragmento por vez. Em um tablet, sempre vai exibir o fragmento que contém a lista de contatos e, conforme for necessário, vai exibir os fragmentos para ver, adicionar e editar contatos no lado direito do aplicativo. Você vai usar o elemento **FragmentManager** e componentes **FragmentManager** para exibir fragmentos dinamicamente. Além disso, vai usar a **pilha de retrocesso** de fragmentos do Android – uma estrutura de dados que armazena fragmentos na ordem “último a entrar, primeiro a sair” (LIFO) – para fornecer suporte automático para o botão *voltar* da barra de sistema do Android e para permitir que o aplicativo remova fragmentos na ordem contrária à qual foram adicionados.

8.3.2 Comunicação de dados entre um fragmento e uma atividade hospedeira

A melhor forma de comunicar dados entre fragmentos e uma atividade hospedeira ou outros fragmentos da atividade é por meio da atividade hospedeira – isso torna os fragmentos mais fáceis de reutilizar, pois não fazem referência entre si diretamente. Normalmente, cada fragmento define uma *interface de métodos de callback* que são implementados na atividade hospedeira. Vamos usar essa técnica para permitir que a atividade **MainActivity** deste aplicativo seja notificada quando o usuário selecionar um contato para exibir, tocar em um item da barra de ação (🔍, ➕ ou ✎), ou terminar de editar um contato já existente ou de adicionar um novo contato.

8.3.3 Método **onSaveInstanceState**

O método **onSaveInstanceState** é chamado pelo sistema quando a configuração do dispositivo muda durante a execução do aplicativo – por exemplo, quando o usuário gira o dispositivo ou abre o teclado em um dispositivo com teclado físico. Esse método pode ser usado para salvar informações de estado que você gostaria de restaurar quando o método `onCreate` do aplicativo fosse chamado como parte da alteração de configuração. Quando um aplicativo é simplesmente colocado em segundo plano, talvez para que o usuário possa atender a uma ligação telefônica ou iniciar outro aplicativo, os componentes da interface gráfica do aplicativo salvam seus conteúdos automaticamente para quando o aplicativo for novamente trazido para o primeiro plano (desde que o sistema não encerre o aplicativo). Usamos **onSaveInstanceState** na Fig. 8.47.

8.3.4 Definindo estilos e aplicando-os nos componentes da interface gráfica do usuário

Você pode definir pares atributo-valor comuns de componentes da interface gráfica do usuário como **recursos style** (Seção 8.4.4). Então, pode aplicar os estilos em todos os componentes que compartilham esses valores (Seção 8.4.7) usando o **atributo style**. Todas as alterações subsequentes feitas em um componente **style** são aplicadas automaticamente a todos os componentes da interface gráfica que o utilizam. Usamos isso para estilizar os componentes **TextView** que exibem as informações de um contato.

8.3.5 Especificação de um fundo para um componente **TextView**

Por padrão, os componentes **TextView** não têm borda. Para definir uma, você pode especificar **Drawable** como o valor do atributo **android:background** de **TextView**. **Drawable** pode ser uma imagem, mas neste aplicativo você vai definir um **Drawable** como um objeto **shape** em um arquivo de recurso (Seção 8.4.5). O arquivo de recurso desse **Drawable** é definido em uma ou mais das pastas **drawable** do aplicativo – neste aplicativo, **textview_border.xml** é definido na pasta **drawable-mdpi**.

8.3.6 Extensão da classe **ListFragment** para criar um fragmento contendo um componente **ListView**

Quando a tarefa principal de um fragmento é exibir uma lista de itens rolante, você pode estender a classe **ListFragment** (pacote **android.app**, Seção 8.6) – é quase idêntico a estender **ListActivity**, como você fez no Capítulo 4. Um componente **ListFragment** usa um elemento **ListView** como layout padrão. Neste aplicativo, em vez de **ArrayAdpater**, vamos usar um objeto **CursorAdapter** (pacote **android.widget**) para exibir os resultados de uma consulta de banco de dados no componente **ListView**.

8.3.7 Manipulação de um banco de dados **SQLite**

As informações de contato são armazenadas em um banco de dados **SQLite**. De acordo com o site www.sqlite.org, **SQLite** é um dos mecanismos de banco de dados mais amplamente distribuídos do mundo. Cada fragmento deste aplicativo interage com um banco de dados **SQLite** por intermédio da classe utilitária **DatabaseConnector** (Seção 8.9). Essa classe usa uma subclasse aninhada de **SQLiteOpenHelper** (pacote **android.database.sqlite**), a qual simplifica a criação do banco de dados e permite obter um objeto **SQLiteDatabase** (pacote **android.database.sqlite**) para manipular o conteúdo de um banco de dados. As consultas ao banco de dados são feitas com **SQL** (**Structured Query Language**), e os resultados da consulta são gerenciados por meio de um objeto **Cursor** (pacote **android.database**).

8.3.8 Execução de operações de banco de dados fora da thread da interface gráfica do usuário com elementos **AsyncTask**

Você deve efetuar *operações longas* ou operações que *bloqueiam* a execução até que terminem (por exemplo, acesso a arquivos e a bancos de dados) *fora* da thread da interface gráfica do usuário. Isso ajuda a manter a velocidade de resposta do aplicativo e evita *caixas de diálogo Activity Not Responding (ANR)*, que aparecem quando o Android acha que a interface gráfica não está respondendo. Quando precisarmos dos resultados de uma operação de banco de dados na thread da interface gráfica, vamos usar uma subclasse

de `AsyncTask` (pacote `android.os`) para efetuar a operação em uma thread e receber os resultados na thread da interface. Os detalhes da criação e manipulação de threads são tratados pela classe `AsyncTask`, assim como a comunicação dos resultados de `AsyncTask` para a thread da interface gráfica do usuário.

8.4 Construção da interface gráfica do usuário e do arquivo de recursos

Nesta seção, você vai criar os arquivos de código-fonte Java adicionais, os arquivos de recurso e os arquivos de layout da interface gráfica do usuário do aplicativo **Address Book**.

8.4.1 Criação do projeto

Comece criando um novo projeto Android. Especifique os valores a seguir na caixa de diálogo **New Android Project** e, em seguida, pressione **Finish**:

- **Application Name:** Address Book
- **Project Name:** AddressBook
- **Package Name:** com.deitel.addressbook
- **Minimum Required SDK:** API18: Android 4.3
- **Target SDK:** API19: Android 4.4
- **Compile With:** API19: Android 4.4
- **Theme:** Holo Light with Dark Action Bar

No segundo passo de **New Android Application** da caixa de diálogo **New Android Project**, deixe as configurações padrão e pressione **Next >**. No passo **Configure Launcher Icon**, selecione uma imagem de ícone de aplicativo e, então, pressione **Next >**. No passo **Create Activity**, selecione **Blank Activity** e pressione **Next >**. No passo **Blank Activity**, deixe as configurações padrão e clique em **Finish** para criar o projeto. Abra `activity_main.xml` no editor **Graphical Layout** e selecione **Nexus 4** na lista suspensa de tipo de tela. Mais uma vez, usaremos esse dispositivo como base para o nosso projeto.

8.4.2 Criação das classes do aplicativo

O aplicativo consiste em cinco classes:

- A classe `MainActivity` (Seção 8.5) gerencia os fragmentos do aplicativo e coordena as interações entre eles.
- A classe `ContactListFragment` (Seção 8.6) é uma subclasse de `ListFragment` que exibe os nomes dos contatos e fornece um item de menu para adicionar um novo contato.
- A classe `AddEditFragment` (Seção 8.7) é uma subclasse de `Fragment` que fornece uma interface gráfica de usuário para adicionar um novo contato ou editar um já existente.
- A classe `DetailsFragment` (Seção 8.8) é uma subclasse de `Fragment` que exibe os dados de um contato e fornece itens de menu para editar e excluir esse contato.
- A classe `DatabaseConnector` (Seção 8.9) é uma subclasse de `Object` que gerencia as interações deste aplicativo com um banco de dados `SQLite`.

A classe `MainActivity` é gerada pelo IDE quando um novo projeto é criado. Como foi feito em projetos anteriores, você deve adicionar as outras classes ao pacote `com.deitel.addressbook` do projeto na pasta `src`. Para fazer isso para cada classe, clique com o botão direito do mouse no pacote e selecione **New > Class**; em seguida, especifique o nome da classe e a superclasse.

8.4.3 strings.xml

A Figura 8.6 mostra os nomes dos recursos de `String` deste aplicativo e os valores correspondentes. Clique duas vezes em `strings.xml` na pasta `res/values` a fim de exibir o editor de recursos para criar esses recursos de `String`.

Nome do recurso	Valor
<code>no_contacts</code>	No Contacts
<code>menuitem_add</code>	Add
<code>menuitem_edit</code>	Edit
<code>menuitem_delete</code>	Delete
<code>button_save_contact</code>	Save Contact
<code>hint_name</code>	Name (Required)
<code>hint_email</code>	E-Mail
<code>hint_phone</code>	Phone
<code>hint_street</code>	Street
<code>hint_city</code>	City
<code>hint_state</code>	State
<code>hint_zip</code>	Zip
<code>label_name</code>	Name:
<code>label_email</code>	E-Mail:
<code>label_phone</code>	Phone:
<code>label_street</code>	Street:
<code>label_city</code>	City:
<code>label_state</code>	State:
<code>label_zip</code>	Zip:
<code>confirm_title</code>	Are You Sure?
<code>confirm_message</code>	This will permanently delete the contact
<code>ok</code>	OK
<code>error_message</code>	You must enter a contact name
<code>button_cancel</code>	Cancel
<code>button_delete</code>	Delete

Figura 8.6 Recursos de `String` usados no aplicativo Address Book.

8.4.4 styles.xml

Nesta seção, você vai definir os estilos dos componentes `TextView` de `DetailsFragment` que exibem as informações de um contato (Seção 8.4.7). Assim como outros recursos, os recursos de estilo são colocados na pasta `res/values` do aplicativo. Quando um projeto é criado, o IDE gera um arquivo `styles.xml` contendo estilos predefinidos. Cada novo estilo criado especifica um nome que é usado para aplicar esse estilo nos componentes

da interface gráfica do usuário e em um ou mais itens, especificando-se os valores de propriedade a serem aplicados. Para criar os novos estilos:

1. Na pasta `res/values` do aplicativo, abra o arquivo `styles.xml` e certifique-se de que a guia **Resources** esteja selecionada na parte inferior da janela do editor.
2. Clique em **Add...**, selecione **Style/Theme** e clique em **OK** para criar um novo estilo.
3. Configure o campo **Name** do estilo como `ContactLabelTextView` e salve o arquivo.
4. Com o estilo `ContactLabelTextView` selecionado, clique em **Add...** e, então, clique em **OK** para adicionar um item ao estilo. Configure os atributos **Name** e **Value** do novo **Item** e salve o arquivo. Repita esse passo para cada nome e valor na Fig. 8.7.

Nome	Valor
<code>android:layout_width</code>	<code>wrap_content</code>
<code>android:layout_height</code>	<code>wrap_content</code>
<code>android:layout_gravity</code>	<code>right center_vertical</code>

Figura 8.7 Atributos de estilo de `ContactLabelTextView`.

5. Repita os passos 2 e 3 para criar um estilo chamado `ContactTextView` – quando clicar em **Add...**, será necessário selecionar **Create a new element at the top level in Resources**. Então, repita o passo 4 para cada nome e valor na Fig. 8.8. Quando terminar, salve e feche `styles.xml`.

Nome	Valor
<code>android:layout_width</code>	<code>wrap_content</code>
<code>android:layout_height</code>	<code>wrap_content</code>
<code>android:layout_gravity</code>	<code>fill_horizontal</code>
<code>android:textSize</code>	<code>16sp</code>
<code>android:background</code>	<code>@drawable/textview_border</code>

Figura 8.8 Atributos de estilo de `ContactTextView`.

8.4.5 `textview_border.xml`

O estilo `ContactTextView` que você criou na seção anterior define a aparência dos componentes `TextView` utilizados para exibir os detalhes de um contato. Você especificou um `Drawable` (isto é, uma imagem ou um elemento gráfico) chamado `@drawable/textview_border` como valor para o atributo `android:background` dos componentes `TextView`. Nesta seção, você vai definir esse `Drawable` na pasta `res/drawable-mdpi` do aplicativo. Se um `Drawable` for definido apenas em uma das pastas `drawable` do projeto, o Android o utilizará em *todos* os tamanhos e resoluções de dispositivo. Para definir o `Drawable`:

1. Clique com o botão direito do mouse na pasta `res/drawable-mdpi` e selecione **New > Android XML File**.
2. Especifique `textview_border.xml` como nome para **File**, selecione **shape** como elemento raiz e, em seguida, clique em **Finish**.
3. Quando este livro estava sendo produzido, o IDE não fornecia um editor para criar elementos `Drawable`; portanto insira o código XML da Fig. 8.9 no arquivo.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <shape xmlns:android="http://schemas.android.com/apk/res/android"
3     android:shape="rectangle" >
4     <corners android:radius="5dp"/>
5     <stroke android:width="1dp" android:color="#555"/>
6     <padding android:top="10dp" android:left="10dp" android:right="10dp"
7         android:bottom="10dp"/>
8 </shape>

```

Figura 8.9 Representação em XML de um Drawable utilizado para colocar uma borda em um componente TextView.

O atributo `android:shape` do elemento **shape** (linha 3) pode ter o valor “rectangle” (usado neste exemplo), “oval”, “line” ou “ring”. O elemento **corners** (linha 4) especifica o raio do canto do retângulo, o que arredonda os cantos. O elemento **stroke** (linha 5) define a largura e a cor da linha do retângulo. O elemento **padding** (linhas 6 e 7) especifica o espaçamento em torno do conteúdo no elemento em que Drawable é aplicado. Você deve especificar separadamente os valores de preenchimento (padding) superior, esquerdo, direito e inferior. Os detalhes completos da definição de formas podem ser vistos em:

<http://developer.android.com/guide/topics/resources/drawable-resource.html#Shape>

8.4.6 Layout de MainActivity: `activity_main.xml`

Você vai fornecer dois layouts para MainActivity – um para dispositivos do tamanho de telefone na pasta `res/layout` e outro para dispositivos do tamanho de tablet na pasta `res/layout-large`. Você precisará adicionar a pasta `layout-large`.

Layout para telefones: `activity_main.xml` em `res/layout`

Para o layout de telefone, abra `activity_main.xml` na pasta `res/layout`. Configure a propriedade `id` do `FrameLayout` como `@id/fragmentContainer`. Esse `FrameLayout` vai ser usado em telefones para exibir os fragmentos do aplicativo. Configure as propriedades `Padding Left`, `Padding Right`, `Padding Top` e `Padding Bottom` para o `FrameLayout` como fez em outros layouts dos capítulos anteriores.

Layout para tablets: `activity_main.xml` em `res/layout-large`

Para o layout de tablet, crie um novo layout `activity_main.xml` na pasta `res/layout-large`. Esse layout deve usar um componente `LinearLayout` horizontal contendo um elemento `ContactListFragment` e um elemento `FrameLayout` vazio. Use as técnicas que aprendeu na Seção 5.4.9 para adicionar o elemento `ContactListFragment` ao layout e, então, adicione o elemento `FrameLayout`. Configure as seguintes propriedades:

- Para o elemento `LinearLayout`, configure **Weight Sum** como 3 – isso ajudará a alocar o espaço horizontal para os elementos `ContactListFragment` e `FrameLayout`.
- Para o fragmento, configure a propriedade `id` como `@+id/contactListFragment`, **Width** como 0, **Height** como `match_parent`, **Weight** como 1 e a margem **Right** como `@dimen/activity_horizontal_margin`.
- Para o elemento `FrameLayout`, configure a propriedade `id` como `@+id/rightPaneContainer`, **Width** como 0, **Height** como `match_parent` e **Weight** como 2.

Configurar a propriedade **Weight Sum** de `LinearLayout` como 3 e, então, configurar as propriedades **Weight** de `ContactListFragment` e `FrameLayout` como 1 e 2, respectivamente, indica que o elemento `ContactListFragment` deve ocupar um terço da largura de `LinearLayout` e que o elemento `FrameLayout` deve ocupar os dois terços restantes.

8.4.7 Layout de DetailsFragment: fragment_details.xml

Quando o usuário toca em um contato no componente MainActivity, o aplicativo exibe o elemento DetailsFragment (Fig. 8.10). O layout desse fragmento (fragment_details.xml) consiste em um componente ScrollView contendo um elemento GridLayout vertical com duas colunas de componentes TextView. Um componente **ScrollView** é um elemento **ViewGroup** que pode conter outros componentes View (como um layout) e que permite aos usuários *rolarem* por conteúdo grande demais para ser exibido na tela. Usamos um componente ScrollView aqui para garantir que o usuário possa rolar pelos detalhes de um contato, caso o dispositivo não tenha espaço vertical suficiente para mostrar todos os componentes TextView da Fig. 8.10. Siga os passos da Seção 5.4.8 para criar o arquivo fragment_details.xml, mas use um componente ScrollView como **Root Element**. Depois de criar o arquivo, configure a propriedade **id** de ScrollView como `@+id/detailsScrollView` e adicione um elemento GridLayout ao componente ScrollView.

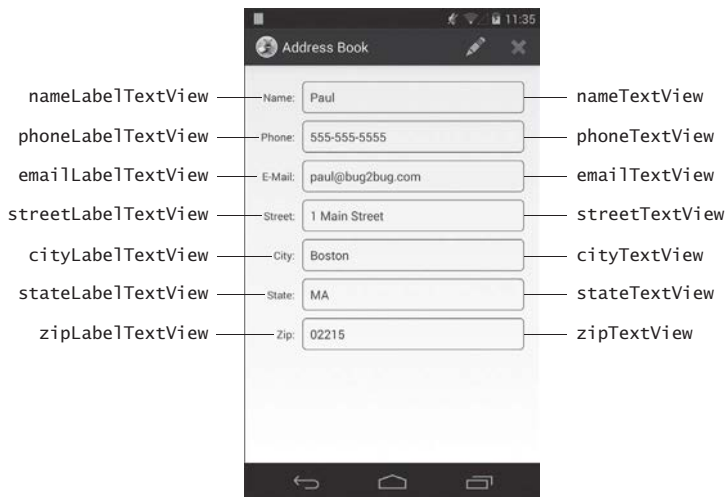


Figura 8.10 Componentes da interface gráfica do usuário de DetailsFragment rotulados com seus valores de propriedade `id`.

Configurações de GridLayout

Para GridLayout, configuramos **Width** como `match_parent`, **Height** como `wrap_content`, **Column Count** como 2 e **Use Default Margins** como `true`. O valor de **Height** permite que o componente ScrollView pai determine a altura real do elemento GridLayout e decida se vai fornecer rolagem. Adicione componentes TextView ao elemento GridLayout conforme mostrado na Fig. 8.10.

Configurações de TextView da coluna da esquerda

Para cada componente TextView na coluna da esquerda, configure sua propriedade `id` conforme especificado na Fig. 8.10 e configure:

- **Row** com um valor de 0 a 6, dependendo da linha.
- **Column** como 0.
- **Text** como o recurso String apropriado de `strings.xml`.

- **Style** (localizada na categoria **View**) como `@style/ContactLabelTextView` — os recursos de estilo são especificados com a sintaxe `@style/nomeDoEstilo`

Configurações de **TextView** da coluna da direita

Para cada componente **TextView** na coluna da direita, configure sua propriedade **Id** conforme especificado na Fig. 8.10 e configure:

- **Row** com um valor de 0 a 6, dependendo da linha.
- **Column** como 1.
- **Style** (localizada na categoria **View**) como `@style/ContactTextView`.

8.4.8 Layout de **AddEditFragment**: `fragment_add_edit.xml`

Quando o usuário toca nos itens  ou  da barra de ação, **MainActivity** exibe o objeto **AddEditFragment** (Fig. 8.11) com um layout (`fragment_add_edit.xml`) que usa um componente **ScrollView** que contém um elemento **GridLayout** vertical de uma coluna. Certifique-se de configurar a propriedade **Id** de **ScrollView** como `@+id/addEditScrollView`. Se for exibido o objeto **AddEditFragment** para adicionar um novo contato, os componentes **EditText** estarão vazios e exibirão *dicas* (Fig. 8.4). Caso contrário, exibirão os dados do contato que foram passados para **AddEditFragment** por **MainActivity**. Cada componente **EditText** especifica as propriedades **Input Type** e **IME Options**. Para dispositivos que exibem um teclado virtual, **Input Type** especifica o teclado a ser exibido quando o usuário tocar no componente **EditText** correspondente. Isso nos permite *personalizar o teclado* para o tipo de dados específico que o usuário deve digitar em determinado componente **EditText**. Usamos a propriedade **IME Options** para exibir um botão **Next** nos teclados virtuais dos componentes `nameEditText`, `emailEditText`, `phoneEditText`, `streetEditText`, `cityEditText` e `stateEditText`. Quando um deles tem o foco, tocar nesse botão transfere o foco para o próximo componente **EditText**. Se o componente `zipEditText` tiver o foco, você pode ocultar o teclado virtual tocando no botão **Done** do teclado.

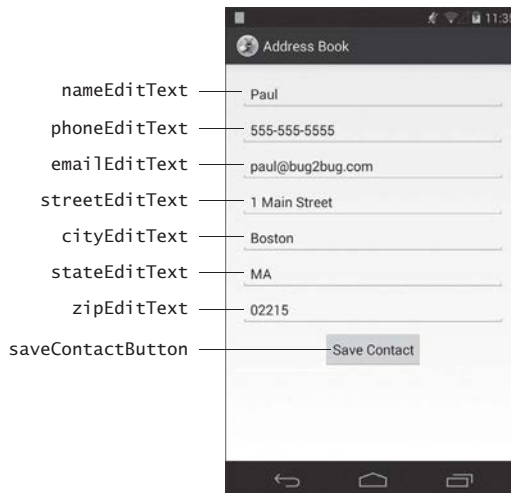


Figura 8.11 Componentes da interface gráfica do usuário de **AddEditFragment** rotulados com seus valores de propriedade **id**. O componente raiz dessa interface gráfica do usuário é um **ScrollView** que contém um elemento **GridLayout** vertical.

Configurações de *GridLayout*

Para *GridLayout*, configuramos **Width** como `match_parent`, **Height** como `wrap_content`, **Column Count** como 1 e **Use Default Margins** como `true`. Adicione os componentes mostrados na Fig. 8.11.

Configurações de *EditText*

Para cada componente *EditText*, configure a propriedade **Id** do elemento *TextView* conforme especificado na Fig. 8.11 e configure:

- **Width** como `match_parent`.
- **Height** como `wrap_content`.
- **Hint** como o recurso de *String* apropriado de `strings.xml`.
- **IME Options** como `actionNext` para todos os componentes *EditText*, exceto `zipEditText`, o qual deve ter o valor `actionDone`.
- **Style** (localizada na categoria **View**) como `@style/ContactLabelTextView` – os recursos de estilo são especificados com a sintaxe `@style/nomeDoEstilo`.

Configure as propriedades **Input Type** dos componentes *EditText* para exibir os teclados apropriados, como segue:

- `nameEditText`: `textPersonName|textCapWords` – para inserir nomes e iniciar cada palavra com uma letra maiúscula.
- `phoneEditText`: `phone` – para inserir números de telefone.
- `emailEditText`: `textEmailAddress` – para inserir um endereço de e-mail.
- `streetEditText`: `textPostalAddress|textCapWords` – para inserir um endereço e iniciar cada palavra com uma letra maiúscula.
- `cityEditText`: `textPostalAddress|textCapWords`.
- `stateEditText`: `textPostalAddress|textCapCharacters` – garante que as abreviações dos estados sejam exibidas em letras maiúsculas.
- `zipEditText`: `number` – para inserir números.

8.4.9 Definição dos menus dos fragmentos

Agora, você vai usar as técnicas que aprendeu na Seção 7.3.4 para criar dois arquivos de recurso de menu na pasta `res/menu` do aplicativo:

- `fragment_contact_list_menu.xml` define o item de menu para adicionar um contato.
- `fragment_details_menu.xml` define os itens de menu para editar um contato já existente e excluir um contato.

Quando os componentes *ContactListFragment* e *DetailsFragment* são exibidos simultaneamente em um tablet, todos os itens de menu aparecem.

As Figuras 8.12 e 8.13 mostram as configurações para os itens de menu nos dois arquivos de recurso de menu. Os valores de **Order in category** de cada item de menu determinam a ordem na qual os itens aparecem na barra de ação. Para o valor de **Icon** de cada item de menu, especificamos um ícone padrão do Android. Você pode ver o conjunto de ícones padrão completo na pasta `platforms` do SDK do Android, sob a pasta `data/res/drawable-hdpi` de cada versão da plataforma. Para fazer referência a esses ícones em seus menus ou layouts, prefixe-os com `@android:drawable/nome_do_icone`.

Nome	Valor
Id	@id/action_add
Order in category	0
Title	@string/menuitem_add
Icon	@android:drawable/ic_menu_add
Show as action	ifRoom withText

Figura 8.12 Item de menu para fragment_contact_list_menu.xml.

Nome	Valor
<i>Item do menu de edição</i>	
Id	@id/action_edit
Order in category	1
Title	@string/menuitem_edit
Icon	@android:drawable/ic_menu_edit
Show as action	ifRoom withText
<i>Item do menu de exclusão</i>	
Id	@id/action_delete
Order in category	2
Title	@string/menuitem_delete
Icon	@android:drawable/ic_delete
Show as action	ifRoom withText

Figura 8.13 Item de menu para fragment_details_menu.xml.

8.5 Classe MainActivity

A classe MainActivity (Figs. 8.14 a 8.23) gerencia os fragmentos do aplicativo e coordena as interações entre eles. Em telefones, MainActivity exibe um fragmento por vez, começando com ContactListFragment. Em tablets, MainActivity sempre exibe ContactListFragment à esquerda do layout e, dependendo do contexto, exibe DetailsFragment ou AddEditFragment nos dois terços à direita do layout.

Instrução package, instruções import e campos de MainActivity

A classe MainActivity (Fig. 8.14) usa a classe FragmentTransaction (importada na linha 6) para adicionar e remover os fragmentos do aplicativo. MainActivity implementa três interfaces:

- ContactListFragment.ContactListFragmentListener contém métodos de callback utilizados por ContactListFragment para informar a MainActivity quando o usuário seleciona um contato na lista de contatos ou adiciona um novo contato.
- DetailsFragment.DetailsFragmentListener contém métodos de callback utilizados por DetailsFragment para informar a MainActivity quando o usuário exclui um contato ou deseja editar um contato já existente.
- AddEditFragment.AddEditFragmentListener contém métodos de callback utilizados por AddEditFragment para informar a MainActivity quando o usuário termina de adicionar um novo contato ou de editar um já existente.

A constante `ROW_ID` (linha 15) é usada como chave em um par chave-valor que é passado entre `MainActivity` e seus fragmentos. A variável de instância `contactListFragment` (linha 17) é usada para avisar a `ContactListFragment` para que atualize a lista de contatos exibida, após um contato ser adicionado ou excluído.

```

1 // MainActivity.java
2 // Armazena os fragmentos do aplicativo Address Book
3 package com.deitel.addressbook;
4
5 import android.app.Activity;
6 import android.app.FragmentTransaction;
7 import android.os.Bundle;
8
9 public class MainActivity extends Activity
10     implements ContactListFragment.ContactListFragmentListener,
11         DetailsFragment.DetailsFragmentListener,
12         AddEditFragment.AddEditFragmentListener
13 {
14     // chaves para armazenar identificador de linha no Bundle passado a um fragmento
15     public static final String ROW_ID = "row_id";
16
17     ContactListFragment contactListFragment; // exibe a lista de contatos
18

```

Figura 8.14 Instrução package, instruções import e campos de MainActivity.

Método sobrescrito onCreate de MainActivity

O método `onCreate` (Fig. 8.15) infla a interface gráfica do usuário de `MainActivity` e, se o aplicativo estiver sendo executado em um dispositivo do tamanho de um telefone, exibe um elemento `ContactListFragment`. Conforme você vai ver na Seção 8.6, é possível configurar um fragmento para ser mantido entre mudanças de configuração, como quando o usuário gira o dispositivo. Se a atividade está sendo restaurada depois de ser desligada ou recriada a partir de uma mudança de configuração, `savedInstanceState` não será `null`. Nesse caso, simplesmente retornamos (linha 28), pois `ContactListFragment` já existe – em um telefone, ele seria mantido e, em um tablet, faz parte do layout de `MainActivity` que foi inflado na linha 24.

```

19 // exibe ContactListFragment quando MainActivity é carregada
20 @Override
21 protected void onCreate(Bundle savedInstanceState)
22 {
23     super.onCreate(savedInstanceState);
24     setContentView(R.layout.activity_main);
25
26     // retorna se a atividade está sendo restaurada, não precisa recriar a
27     // interface gráfica do usuário
28     if (savedInstanceState != null)
29         return;
30
31     // verifica se o layout contém fragmentContainer (layout para telefone);
32     // ContactListFragment é sempre exibido
33     if (findViewById(R.id.fragmentContainer) != null)
34     {
35         // cria ContactListFragment
36         contactListFragment = new ContactListFragment();
37

```

Figura 8.15 Método sobrescrito `onCreate` de MainActivity.

```

36
37      // adiciona o fragmento a FrameLayout
38      FragmentTransaction transaction =
39          getFragmentManager().beginTransaction();
40      transaction.add(R.id.fragmentContainer, contactListFragment);
41      transaction.commit(); // faz ContactListFragment aparecer
42  }
43  }
44

```

Figura 8.15 Método sobrescrito onCreate de MainActivity.

Se `R.id.fragmentContainer` existe no layout de MainActivity (linha 32), então o aplicativo está sendo executado em um telefone. Nesse caso, a linha 35 cria o elemento `ContactListFragment` e, então, as linhas 38 a 41 usam um componente `FragmentTransaction` para adicionar o `ContactListFragment` à interface do usuário. As linhas 38 e 39 chamam o método **beginTransaction** de `FragmentManager` para obter um elemento `FragmentTransaction`. Em seguida, a linha 40 usa o método **add** de `FragmentTransaction` para especificar que, quando `FragmentTransaction` terminar, `ContactListFragment` deve ser anexado à View com o identificador especificado como primeiro argumento. Por fim, a linha 41 usa o método **commit** de `FragmentTransaction` para finalizar a transação e exibir o elemento `ContactListFragment`.

Método sobrescrito onResume de MainActivity

O método `onResume` (Fig. 8.16) determina se `contactListFragment` é `null` – se for, o aplicativo está sendo executado em um tablet, de modo que as linhas 55 a 57 usam o elemento `FragmentManager` para obter uma referência ao objeto `ContactListFragment` existente no layout de MainActivity.

```

45      // chamado quando MainActivity recomeça
46      @Override
47      protected void onResume()
48      {
49          super.onResume();
50
51          // se contactListFragment é null, a atividade está sendo executada em tablet;
52          // portanto, obtém referência a partir de FragmentManager
53          if (contactListFragment == null)
54          {
55              contactListFragment =
56                  (ContactListFragment) getFragmentManager().findFragmentById(
57                      R.id.contactListFragment);
58          }
59      }
60

```

Figura 8.16 Método sobrescrito onResume de MainActivity.

Método onContactSelected de MainActivity

O método `onContactSelected` (Fig. 8.17) da interface `ContactListFragment.ContactListFragmentListener` é chamado por `ContactListFragment` para notificar a MainActivity quando o usuário seleciona um contato para exibir. Se o aplicativo está sendo executado em um telefone (linha 65), a linha 66 chama o método `displayContact` (Fig. 8.18), o qual substitui o componente `ContactListFragment` no elemento `fragmentContainer` (definido na Seção 8.4.6) pelo componente `DetailsFragment` que

mostra as informações do contato. Em um tablet, a linha 69 chama o método `popBackStack` de `FragmentManager` para *desempilhar* (remover) o fragmento superior da pilha de retrocesso e, então, a linha 70 chama `displayContact`, que substitui o conteúdo de `rightPaneContainer` (definido na Seção 8.4.6) pelo componente `DetailsFragment` que mostra as informações do contato.

```

61 // exibe DetailsFragment do contato selecionado
62 @Override
63 public void onContactSelected(long rowID)
64 {
65     if (findViewById(R.id.fragmentContainer) != null) // telefone
66         displayContact(rowID, R.id.fragmentContainer);
67     else // tablet
68     {
69         getFragmentManager().popBackStack(); // remove o topo da pilha de retrocesso
70         displayContact(rowID, R.id.rightPaneContainer);
71     }
72 }
73

```

Figura 8.17 Método `onContactSelected` de `MainActivity`.

Método `displayContact` de `MainActivity`

O método `displayContact` (Fig. 8.18) cria o componente `DetailsFragment` que exibe o contato selecionado e usa um elemento `FragmentTransaction` para anexá-lo na interface gráfica do usuário. Você pode passar argumentos para um fragmento colocando-os em um objeto `Bundle` de pares chave-valor – fazemos isso para passar o elemento `rowID` do contato selecionado, a fim de que o componente `DetailsFragment` saiba qual contato deve obter do banco de dados. A linha 80 cria o objeto `Bundle`. A linha 81 chama seu método `putLong` para armazenar um par chave-valor contendo o objeto `ROW_ID` (uma `String`) como chave e `rowID` (um valor `long`) como valor. A linha 82 passa o objeto `Bundle` para o método `setArguments` de `Fragment` – o fragmento pode então extrair as informações do objeto `Bundle` (como você vai ver na Seção 8.8). As linhas 85 e 86 obtêm um elemento `FragmentTransaction` e, então, a linha 87 chama o método `replace` de `FragmentTransaction` para especificar que, quando o elemento `FragmentTransaction` terminar, o componente `DetailsFragment` deve substituir o conteúdo da `View` pelo identificador especificado como primeiro argumento. A linha 88 chama o método `addToBackStack` de `FragmentTransaction` para *empilhar* (adicionar) o componente `DetailsFragment` na pilha de retrocesso. Isso permite que o usuário toque no botão *voltar* para extrair o fragmento da pilha e permite que `MainActivity` extraia o fragmento dela via programação.

```

74 // exibe um contato
75 private void displayContact(long rowID, int viewID)
76 {
77     DetailsFragment detailsFragment = new DetailsFragment();
78
79     // especifica rowID como argumento para DetailsFragment
80     Bundle arguments = new Bundle();
81     arguments.putLong(ROW_ID, rowID);
82     detailsFragment.setArguments(arguments);
83

```

Figura 8.18 Método `displayContact` de `MainActivity`.

```

84 // usa um elemento FragmentTransaction para exibir o componente DetailsFragment
85 FragmentTransaction transaction =
86     getFragmentManager().beginTransaction();
87 transaction.replace(viewID, detailsFragment);
88 transaction.addToBackStack(null);
89 transaction.commit(); // faz DetailsFragment aparecer
90 }
91

```

Figura 8.18 Método `displayContact` de `MainActivity`.

Método `onAddContact` de `MainActivity`

O método `onAddContact` (Fig. 8.19) da interface `ContactListFragment`. `ContactListFragmentListener` é chamado por `ContactListFragment` para notificar a `MainActivity` quando o usuário opta por adicionar um novo contato. Se o layout contém o objeto `fragmentContainer`, a linha 97 chama `displayAddEditFragment` (Fig. 8.20) para exibir o componente `AddEditFragment` no `fragmentContainer`; caso contrário, a linha 99 chama `displayAddEditFragment` para exibir o fragmento no elemento `rightPaneContainer`. O segundo argumento é um objeto `Bundle`. Especificar `null` indica que um novo contato está sendo adicionado.

```

92 // exibe o componente AddEditFragment para adicionar um novo contato
93 @Override
94 public void onAddContact()
95 {
96     if (findViewById(R.id.fragmentContainer) != null) // telefone
97         displayAddEditFragment(R.id.fragmentContainer, null);
98     else // tablet
99         displayAddEditFragment(R.id.rightPaneContainer, null);
100 }
101

```

Figura 8.19 Método `onAddContact` de `MainActivity`.

Método `displayAddEditFragment` de `MainActivity`

O método `displayAddEditFragment` (Fig. 8.20) recebe o identificador de recurso de uma `View` especificando onde anexar o componente `AddEditFragment` e um objeto `Bundle` de pares chave-valor. Se o segundo argumento é `null`, um novo contato está sendo adicionado; caso contrário, o objeto `Bundle` contém os dados a exibir no componente `AddEditFragment` para edição. A linha 105 cria o componente `AddEditFragment`. Se o argumento de `Bundle` não é `null`, a linha 108 o utiliza para configurar os argumentos de `Fragment`. Então, as linhas 111 a 115 criam o componente `FragmentTransaction`, substituem o conteúdo da `View` pelo identificador de recurso especificado, adicionam o fragmento à pilha de retrocesso e efetivam a transação.

```

102 // exibe fragmento para adicionar um novo contato ou editar um já existente
103 private void displayAddEditFragment(int viewID, Bundle arguments)
104 {
105     AddEditFragment addEditFragment = new AddEditFragment();
106
107     if (arguments != null) // editando contato existente
108         addEditFragment.setArguments(arguments);

```

Figura 8.20 Método `displayAddEditContact` de `MainActivity`. (continua)

```
109
110 // usa um elemento FragmentTransaction para exibir o componente AddEditFragment
111 FragmentTransaction transaction =
112     getFragmentManager().beginTransaction();
113 transaction.replace(viewID, addEditFragment);
114 transaction.addToBackStack(null);
115 transaction.commit(); // faz AddEditFragment aparecer
116 }
117
```

Figura 8.20 Método `displayAddEditContact` de `MainActivity`.

Método `onContactDeleted` de `MainActivity`

O método `onContactDeleted` (Fig. 8.21) da interface `DetailsFragment.DetailsFragmentManagerListener` é chamado por `DetailsFragment` para notificar a `MainActivity` quando o usuário exclui um contato. Nesse caso, a linha 122 extrai o elemento `DetailsFragment` da pilha de retrocesso. Se o aplicativo está sendo executado em um tablet, a linha 125 chama o método `updateContactList` de `contactListFragment` para recarregar os contatos.

```
118 // retorna à lista de contatos quando exibiu contato excluído
119 @Override
120 public void onContactDeleted()
121 {
122     getFragmentManager().popBackStack(); // remove o topo da pilha de retrocesso
123
124     if (findViewById(R.id.fragmentContainer) == null) // tablet
125         contactListFragment.updateContactList();
126 }
127
```

Figura 8.21 Método `onContactDeleted` de `MainActivity`.

Método `onEditContact` de `MainActivity`

O método `onEditContact` (Fig. 8.22) da interface `DetailsFragment.DetailsFragmentManagerListener` é chamado por `DetailsFragment` para notificar a `MainActivity` quando o usuário toca no item de menu para editar um contato. O elemento `DetailsFragment` passa um objeto `Bundle` contendo os dados do contato, para que possam ser exibidos nos componentes `EditText` de `AddEditFragment` para edição. Se o layout contém o objeto `fragmentContainer`, a linha 133 chama `displayAddEditFragment` para exibir o componente `AddEditFragment` no `fragmentContainer`; caso contrário, a linha 135 chama `displayAddEditFragment` para exibir o componente `AddEditFragment` no elemento `rightPaneContainer`.

```
128 // exibe o componente AddEditFragment para editar um contato já existente
129 @Override
130 public void onEditContact(Bundle arguments)
131 {
132     if (findViewById(R.id.fragmentContainer) != null) // telefone
133         displayAddEditFragment(R.id.fragmentContainer, arguments);
134     else // tablet
135         displayAddEditFragment(R.id.rightPaneContainer, arguments);
136 }
137
```

Figura 8.22 Método `onEditContact` de `MainActivity`.

Método `onAddEditCompleted` de `MainActivity`

O método `onAddEditCompleted` (Fig. 8.23) da interface `AddEditFragment.AddEditFragmentListener` é chamado por `AddEditFragment` para notificar a `MainActivity` quando o usuário salva um novo contato ou salva alterações feitas em um contato já existente. A linha 142 extrai o elemento `AddEditFragment` da pilha de retrocesso. Se o aplicativo está sendo executado em um tablet (linha 144), a linha 146 extrai o topo da pilha de retrocesso novamente para remover o componente `DetailsFragment` (se houver um). Então, a linha 147 atualiza a lista de contatos no componente `ContactListFragment`, e a linha 150 exibe os detalhes do contato novo ou atualizado no elemento `rightPaneContainer`.

```

138 // atualiza a interface gráfica do usuário após um contato novo ou atualizado
139 // ser salvo
140 @Override
141 public void onAddEditCompleted(long rowID)
142 {
143     getFragmentManager().popBackStack(); // remove o topo da pilha de retrocesso
144
145     if (findViewById(R.id.fragmentContainer) == null) // tablet
146     {
147         getFragmentManager().popBackStack(); // remove o topo da pilha de retrocesso
148         contactListFragment.updateContactList(); // atualiza os contatos
149
150         // em tablet, exibe o contato que acabou de ser adicionado ou editado
151         displayContact(rowID, R.id.rightPaneContainer);
152     }
153 }
```

Figura 8.23 Método `onAddEditCompleted` de `MainActivity`.

8.6 Classe `ContactListFragment`

A classe `ContactListFragment` (Figs. 8.24 a 8.33) estende `ListFragment` para exibir a lista de contatos em um componente `ListView` e fornece um item de menu para adicionar um novo contato.

A instrução `package` e as instruções `import` de `ContactListFragment`

A Figura 8.24 lista a instrução `package` e as instruções `import` de `ContactListFragment`. Realçamos as instruções `import` das novas classes e interfaces.

```

1 // ContactListFragment.java
2 // Exibe a lista de nomes de contato
3 package com.deitel.addressbook;
4
5 import android.app.Activity;
6 import android.app.ListFragment;
7 import android.database.Cursor;
8 import android.os.AsyncTask;
9 import android.os.Bundle;
10 import android.view.Menu;
11 import android.view.MenuInflater;
```

Figura 8.24 Instrução `package` e instruções `import` de `ContactListFragment`. (continua)

```

12 import android.view.MenuItem;
13 import android.view.View;
14 import android.widget.AdapterView;
15 import android.widget.AdapterView.OnItemClickListener;
16 import android.widget.CursorAdapter;
17 import android.widget.ListView;
18 import android.widget.SimpleCursorAdapter;
19

```

Figura 8.24 Instrução package e instruções import de ContactListFragment.

Interface ContactListFragmentListener e variáveis de instância de ContactListFragment

A Figura 8.25 inicia a declaração da classe ContactListFragment. As linhas 23 a 30 declaram a interface aninhada ContactListFragmentListener, a qual contém os métodos de callback implementados por MainActivity para ser notificada quando o usuário selecionar um contato (linha 26) e quando tocar no item de menu para adicionar um novo contato (linha 29). A linha 32 declara a variável de instância listener, a qual vai fazer referência ao objeto (MainActivity) que implementa a interface. A variável de instância contactListView (linha 34) vai se referir ao componente ListView interno de ContactListFragment para que possamos interagir com ele via programação. A variável de instância contactAdapter vai se referir ao CursorAdapter que preenche o componente ListView de AddressBook.

```

20 public class ContactListFragment extends ListFragment
21 {
22     // métodos de callback implementados por MainActivity
23     public interface ContactListFragmentListener
24     {
25         // chamado quando o usuário seleciona um contato
26         public void onContactSelected(long rowID);
27
28         // chamado quando o usuário decide adicionar um contato
29         public void onAddContact();
30     }
31
32     private ContactListFragmentListener listener;
33
34     private ListView contactListView; // ListView de ListActivity
35     private CursorAdapter contactAdapter; // adaptador para ListView
36

```

Figura 8.25 Interface ContactListFragmentListener e variáveis de instância de ContactListFragment.

Métodos sobrescritos onAttach e onDetach de ContactListFragment

A classe ContactListFragment sobrepõe os métodos de ciclo de vida onAttach e onDetach de Fragment (Fig. 8.26) para configurar a variável de instância listener. Neste aplicativo, listener se refere à atividade hospedeira (linha 42) quando o elemento ContactListFragment é anexado, e é configurado como null (linha 50) quando o elemento ContactListFragment é desanexado.

```

37 // configura ContactListFragmentListener quando o fragmento é anexado
38 @Override
39 public void onAttach(Activity activity)
40 {
41     super.onAttach(activity);
42     listener = (ContactListFragmentListener) activity;
43 }
44
45 // remove ContactListFragmentListener quando o fragmento é desanexado
46 @Override
47 public void onDetach()
48 {
49     super.onDetach();
50     listener = null;
51 }
52

```

Figura 8.26 Métodos sobrescritos onAttach e onDetach de ContactListFragment.

Método sobrescrito onCreateView de ContactListFragment

Lembre-se de que a classe ListFragment já contém um componente ListView; portanto, não precisamos inflar a interface gráfica do usuário, como nos fragmentos do aplicativo anterior. Contudo, a classe ContactListFragment tem tarefas que devem ser executadas depois que seu layout padrão for inflado. Por isso, ContactListFragment sobrepõe o método de ciclo de vida **onViewCreated** de Fragment (Fig. 8.27), o qual é chamado após onCreateView.

```

53 // chamado depois que a View é criada
54 @Override
55 public void onCreateView(View view, Bundle savedInstanceState)
56 {
57     super.onCreateView(view, savedInstanceState);
58     setRetainInstance(true); // salva o fragmento entre mudanças de configuração
59     setHasOptionsMenu(true); // este fragmento tem itens de menu a exibir
60
61     // configura o texto a exibir quando não houver contatos
62     setEmptyText(getResources().getString(R.string.no_contacts));
63
64     // obtém referência de ListView e configura ListView
65     contactListView = getListView();
66     contactListView.setOnItemClickListener(viewClickListener);
67     contactListView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
68
69     // mapeia o nome de cada contato em um componente TextView no layout de ListView
70     String[] from = new String[] { "name" };
71     int[] to = new int[] { android.R.id.text1 };
72     contactAdapter = new SimpleCursorAdapter(getActivity(),
73         android.R.layout.simple_list_item_1, null, from, to, 0);
74     setListAdapter(contactAdapter); // configura o adaptador que fornece dados
75 }
76

```

Figura 8.27 Método sobrescrito onCreateView de ContactListFragment.

A linha 58 chama o método **setRetainInstance** de Fragment com o argumento **true** para indicar que o elemento ContactListFragment deve ser mantido, em vez de recriado, quando a atividade hospedeira for recriada em uma mudança de configuração (por exemplo, quando o usuário girar o dispositivo). A linha 59 indica que o elemento

`ContactListFragment` tem itens de menu que devem ser exibidos na barra de ação da atividade (ou em seu menu de opções). O método `setEmptyText` de `ListFragment` (linha 62) especifica o texto a ser exibido ("No Contacts") quando não houver itens no adaptador de `ListView`.

A linha 65 usa o método herdado `getListView` de `ListActivity` a fim de obter uma referência para o componente `ListView` interno. A linha 66 configura o elemento `OnItemClickListener` de `ListView` como `viewContactListener` (Fig. 8.28), o qual responde quando o usuário toca em um contato no componente `ListView`. A linha 67 chama o método `setChoiceMode` de `ListView` para indicar que somente um item pode ser selecionado por vez.

Configurando o elemento `CursorAdapter` que vincula dados do banco de dados ao componente `ListView`

Para exibir os resultados do `Cursor` em um componente `ListView`, criamos um novo objeto `CursorAdapter` (linhas 70 a 73), o qual expõe os dados do `Cursor` de tal maneira que possam ser usados por um componente `ListView`. **`SimpleCursorAdapter`** é uma subclasse de `CursorAdapter` projetada para simplificar o mapeamento de colunas de um `Cursor` diretamente nos componentes `TextView` ou `ImageView` definidos em seus layouts XML. Para criar `SimpleCursorAdapter`, você primeiro define arrays contendo os nomes a serem mapeados nos componentes da interface gráfica do usuário e nos identificadores de recurso dos componentes da interface que vão exibir os dados das colunas nomeadas. A linha 70 cria um array de `Strings` indicando que somente a coluna "name" vai ser exibida, e a linha 71 cria um array de inteiros paralelo, contendo os identificadores de recurso correspondentes dos componentes da interface gráfica. O Capítulo 4 mostrou que você pode criar seus próprios recursos de layout para itens de `ListView`. Neste aplicativo, usamos um recurso de layout predefinido do Android, chamado `android.R.layout.simple_list_item_1` – um layout que contém um componente `TextView` com o identificador `android.R.id.text1`. As linhas 72 e 73 criam `SimpleCursorAdapter`. Seu construtor recebe:

- o objeto `Context` no qual o componente `ListView` está sendo executado (isto é, `MainActivity`).
- o identificador do recurso do layout utilizado para exibir cada item no componente `ListView`.
- o objeto `Cursor` que dá acesso aos dados – fornecemos `null` para esse argumento, porque vamos especificar o objeto `Cursor` posteriormente.
- o array de `Strings` contendo os nomes de coluna a serem exibidos.
- o array de inteiros contendo os identificadores de recurso da interface gráfica correspondentes.
- o último argumento normalmente é 0.

A linha 74 usa o método herdado `setListAdapter` de `ListActivity` para vincular o componente `ListView` ao elemento `CursorAdapter`, para que `ListView` possa exibir os dados.

Componente `viewContactListener` que processa eventos de seleção de item em `ListView`

O componente `viewContactListener` (Fig. 8.28) notifica a `MainActivity` quando o usuário toca em um contato para exibir. A linha 84 passa o argumento `id` – o identificador de linha do contato selecionado – para o método `onContactSelected` de `listener` (Fig. 8.17).

```

77 // responde ao toque do usuário no nome de um contato no componente ListView
78 OnItemClickListener viewClickListener = new OnItemClickListener()
79 {
80     @Override
81     public void onItemClick(AdapterView<?> parent, View view,
82         int position, long id)
83     {
84         listener.onContactSelected(id); // passa a seleção para MainActivity
85     }
86 }; // fim de viewClickListener
87

```

Figura 8.28 Componente viewClickListener que processa eventos de seleção de item em ListView.

Método sobrescrito onResume de ContactListFragment

O método de ciclo de vida onResume de Fragment (Fig. 8.29) cria e executa um objeto AsyncTask (linha 93) do tipo GetContactsTask (definido na Fig. 8.30), que obtém a lista completa de contatos do banco de dados e configura o objeto Cursor de contactAdapter para preencher o componente ListView de ContactListFragment. O método **execute** de AsyncTask executa a tarefa em uma thread separada. O argumento do método execute neste caso indica que a tarefa não recebe argumentos – esse método pode receber um número variável de argumentos que, por sua vez, são passados como argumentos para o método doInBackground da tarefa. Sempre que a linha 93 é executada, ela cria um novo objeto GetContactsTask – isso é obrigatório, pois cada AsyncTask pode ser executada *apenas uma vez*.

```

88 // quando o fragmento recomeça, usa um elemento GetContactsTask para
89 // carregar os contatos
90 @Override
91 public void onResume()
92 {
93     super.onResume();
94     new GetContactsTask().execute((Object[]) null);
95 }
96

```

Figura 8.29 Método sobrescrito onResume de ContactListFragment.

Subclasse GetContactsTask de AsyncTask

A classe aninhada GetContactsTask (Fig. 8.30) estende a classe AsyncTask. A classe define como interagir com o DatabaseConnector (Seção 8.9) para obter os nomes de todos os contatos e retornar os resultados para a thread da interface gráfica do usuário dessa atividade, a fim de que sejam exibidos no componente ListView. AsyncTask é um tipo genérico que exige três parâmetros de tipo:

- O tipo de lista de parâmetros de comprimento variável do método **doInBackground** de AsyncTask (linhas 103 a 108) – quando você chama o método execute da tarefa, doInBackground executa a tarefa em uma thread separada. Especificamos Object como parâmetro de tipo e passamos null como argumento para o método execute de AsyncTask, pois GetContactsTask não exige dados adicionais para executar sua tarefa.
- O tipo de lista de parâmetros de comprimento variável do método **onProgressUpdate** de AsyncTask – esse método é executado na thread da interface gráfica do usuário e

é usado para receber *atualizações intermediárias* do tipo especificado de uma tarefa de execução longa. Não usamos esse recurso neste exemplo, de modo que especificamos o tipo `Object` aqui e ignoramos esse parâmetro de tipo.

- O tipo do resultado da tarefa, o qual é passado para o método `onPostExecute` de `AsyncTask` (linhas 111 a 116) – esse método é executado na thread da interface gráfica do usuário e permite que `ContactListFragment` utilize os resultados de `AsyncTask`.

Uma vantagem importante de usar `AsyncTask` é que esse elemento trata dos detalhes da criação de threads e da execução de seus métodos nas threads apropriadas, de modo que você não precisa interagir com o mecanismo de threads diretamente.

```

96 // executa a consulta de banco de dados fora da thread da interface gráfica
97 // do usuário
98 private class GetContactsTask extends AsyncTask<Object, Object, Cursor>
99 {
100     DatabaseConnector databaseConnector =
101         new DatabaseConnector(getActivity());
102
103     // abre o banco de dados e retorna um Cursor para todos os contatos
104     @Override
105     protected Cursor doInBackground(Object... params)
106     {
107         databaseConnector.open();
108         return databaseConnector.getAllContacts();
109     }
110
111     // usa o Cursor retornado pelo método doInBackground
112     @Override
113     protected void onPostExecute(Cursor result)
114     {
115         contactAdapter.changeCursor(result); // configura o Cursor do adaptador
116         databaseConnector.close();
117     }
118 } // fim da classe GetContactsTask

```

Figura 8.30 Subclasse `GetContactsTask` de `AsyncTask`.

As linhas 99 e 100 criam um novo objeto de nossa classe utilitária `DatabaseConnector`, passando o objeto `Context` (a atividade hospedeira de `ContactListFragment`) como argumento para o construtor da classe. O método `doInBackground` usa `DatabaseConnector` para abrir a conexão de banco de dados e obtém todos os contatos do banco de dados. O objeto `Cursor` retornado por `getAllContacts` é passado para o método `onPostExecute`, o qual recebe o `Cursor` que contém os resultados e o passa para o método `changeCursor` de `contactAdapter`. Isso permite que o componente `ListView` de `ContactListFragment` preencha a si mesmo com os nomes dos contatos.

Método sobrescrito `onStop` de `ContactListFragment`

O método de ciclo de vida `onStop` de `Fragment` (Fig. 8.31) é chamado depois de `onPause`, quando o fragmento não está mais visível para o usuário. Nesse caso, o objeto `Cursor` que nos permite preencher o componente `ListView` não é necessário, de modo que a linha 123 chama o método `getCursor` de `CursorAdapter` para obter o objeto `Cursor` atual a

partir de `contactAdapter`. A linha 124 chama o método `changeCursor` de `CursorAdapter` com o argumento `null` para remover o objeto `Cursor` de `CursorAdapter`. Então, a linha 127 chama o método `close` de `Cursor` para liberar os recursos usados pelo objeto `Cursor`.

```

119 // quando o fragmento para, fecha o Cursor e remove de contactAdapter
120 @Override
121 public void onStop()
122 {
123     Cursor cursor = contactAdapter.getCursor(); // obtém o objeto Cursor atual
124     contactAdapter.changeCursor(null); // agora o adaptador não tem objeto Cursor
125
126     if (cursor != null)
127         cursor.close(); // libera os recursos do Cursor
128
129     super.onStop();
130 }
131
```

Figura 8.31 Método sobrescrito `onStop` de `ContactListFragment`.

Métodos sobrescritos `onCreateOptionsMenu` e `onOptionsItemSelected` de `ContactListFragment`

O método `onCreateOptionsMenu` (Fig. 8.32, linhas 133 a 138) usa seu argumento `MenuInflater` para criar o menu a partir de `fragment_contact_list_menu.xml`, que contém a definição do item de menu para adicionar contatos (⊕). Se o usuário toca nesse componente `MenuItem`, o método `onOptionsItemSelected` (linhas 141 a 152) chama o método `onAddContact` de `listener` para notificar a `MainActivity` de que o usuário quer adicionar um novo contato. Então, `MainActivity` exibe o componente `AddEditFragment` (Seção 8.7).

```

132 // exibe os itens de menu deste fragmento
133 @Override
134 public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
135 {
136     super.onCreateOptionsMenu(menu, inflater);
137     inflater.inflate(R.menu.fragment_contact_list_menu, menu);
138 }
139
140 // trata a escolha no menu de opções
141 @Override
142 public boolean onOptionsItemSelected(MenuItem item)
143 {
144     switch (item.getItemId())
145     {
146         case R.id.action_add:
147             listener.onAddContact();
148             return true;
149     }
150
151     return super.onOptionsItemSelected(item); // chama o método de super
152 }
153
```

Figura 8.32 Métodos sobrescritos `onCreateOptionsMenu` e `onOptionsItemSelected` de `ContactListFragment`.

Método `updateContactList` de `ContactListFragment`

O método `updateContactList` (Fig. 8.33) cria e executa um objeto `GetContactsTask` para atualizar a lista de contatos.

```
154 // atualiza o conjunto de dados
155 public void updateContactList()
156 {
157     new GetContactsTask().execute((Object[]) null);
158 }
159 } // fim da classe ContactListFragment
```

Figura 8.33 Método `updateContactList` de `ContactListFragment`.

8.7 Classe `AddEditFragment`

A classe `AddEditFragment` (Figs. 8.34 a 8.40) fornece a interface para adicionar novos contatos ou editar os já existentes.

A instrução `package` e as instruções `import` de `AddEditFragment`

A Figura 8.34 lista a instrução `package` e as instruções `import` da classe `AddEditFragment`. Nenhuma classe nova é usada nesse fragmento.

```
1 // AddEditFragment.java
2 // Permite ao usuário adicionar um novo contato ou editar um já existente
3 package com.deitel.addressbook;
4
5 import android.app.Activity;
6 import android.app.AlertDialog;
7 import android.app.Dialog;
8 import android.app.DialogFragment;
9 import android.app.Fragment;
10 import android.content.Context;
11 import android.os.AsyncTask;
12 import android.os.Bundle;
13 import android.view.LayoutInflater;
14 import android.view.View;
15 import android.view.View.OnClickListener;
16 import android.view.ViewGroup;
17 import android.view.inputmethod.InputMethodManager;
18 import android.widget.Button;
19 import android.widget.EditText;
20
21 public class AddEditFragment extends Fragment
22 {
```

Figura 8.34 Instrução `package` e instruções `import` de `AddEditFragment`.

Interface `AddEditFragmentListener`

A Figura 8.35 declara a interface aninhada `AddEditFragmentListener` que contém o método de callback `onAddEditCompleted`, implementado por `MainActivity` para ser notificada quando o usuário salva um novo contato ou salva alterações feitas em um já existente.

```

23 // método de callback implementado por MainActivity
24 public interface AddEditFragmentListener
25 {
26     // chamado após a conclusão da edição para que o contato possa ser reexibido
27     public void onAddEditCompleted(long rowID);
28 }
29

```

Figura 8.35 Interface AddEditFragmentListener.

Variáveis de instância de AddEditFragment

A Figura 8.36 lista as variáveis de instância da classe:

- A variável `listener` faz referência ao elemento `AddEditFragmentListener` que é notificado quando o usuário clica no botão **Save Contact**.
- A variável `rowID` representa o contato atual que está sendo manipulado caso esse fragmento tenha sido exibido, a fim de permitir que o usuário edite um contato já existente.
- A variável `contactInfoBundle` será `null` se um novo contato estiver sendo adicionado, ou fará referência a um objeto `Bundle` de informações de contato caso um contato já existente esteja sendo editado.
- As variáveis de instância nas linhas 36 a 42 farão referência aos componentes `EditText` do fragmento.

```

30 private AddEditFragmentListener listener;
31
32 private long rowID; // identificador de linha do contato no banco de dados
33 private Bundle contactInfoBundle; // argumentos para editar um contato
34
35 // componentes EditText para informações de contato
36 private EditText nameEditText;
37 private EditText phoneEditText;
38 private EditText emailEditText;
39 private EditText streetEditText;
40 private EditText cityEditText;
41 private EditText stateEditText;
42 private EditText zipEditText;
43

```

Figura 8.36 Variáveis de instância de AddEditFragment.

Métodos sobrescritos onAttach e onDetach de AddEditFragment

A classe `AddEditFragment` sobrescreve os métodos de ciclo de vida `onAttach` e `onDetach` de `Fragment` (Fig. 8.37) a fim de configurar a variável de instância `listener` para fazer referência à atividade hospedeira (linha 49) quando o elemento `AddEditFragment` é anexado, e para configurar `listener` como `null` (linha 57) quando o elemento é desanexado.

```

44 // configura AddEditFragmentListener quando o fragmento é anexado
45 @Override
46 public void onAttach(Activity activity)
47 {
48     super.onAttach(activity);

```

Figura 8.37 Métodos sobrescritos `onAttach` e `onDetach` de `AddEditFragment`. (continua)

```

49     listener = (AddEditFragmentListener) activity;
50 }
51
52 // remove AddEditFragmentListener quando o fragmento é desanexado
53 @Override
54 public void onDetach()
55 {
56     super.onDetach();
57     listener = null;
58 }
59

```

Figura 8.37 Métodos sobrescritos onAttach e onDetach de AddEditFragment.

Método sobrescrito onCreateView de AddEditFragment

No método onCreateView (Fig. 8.38), as linhas 70 a 78 inflam a interface gráfica do usuário e obtêm os componentes EditText do fragmento. Em seguida, usamos o método getArguments de Fragment para obter o objeto Bundle de argumentos (se houver). Quando ativamos o elemento AddEditFragment a partir de MainActivity, não passamos um objeto Bundle, pois o usuário está adicionando as informações de um novo contato. Nesse caso, getArguments vai retornar null. Se retornar um Bundle (linha 82), então o elemento AddEditFragment foi ativado a partir de DetailsFragment e o usuário optou por editar um contato já existente. As linhas 84 a 91 leem os argumentos desse Bundle chamando os métodos getLong (linha 84) e getString, e os dados String são exibidos nos componentes EditText para edição. As linhas 95 a 97 registram um receptor (Fig. 8.39) para o componente Button **Save Contact**.

```

60 // chamado quando a view do fragmento precisa ser criada
61 @Override
62 public View onCreateView(LayoutInflater inflater, ViewGroup container,
63     Bundle savedInstanceState)
64 {
65     super.onCreateView(inflater, container, savedInstanceState);
66     setRetainInstance(true); // salva o fragmento entre mudanças de configuração
67     setHasOptionsMenu(true); // o fragmento tem itens de menu a exibir
68
69     // infla a interface gráfica do usuário e obtém referências para os componentes
70     // EditText
71     View view =
72         inflater.inflate(R.layout.fragment_add_edit, container, false);
73     nameEditText = (EditText) view.findViewById(R.id.nameEditText);
74     phoneEditText = (EditText) view.findViewById(R.id.phoneEditText);
75     emailEditText = (EditText) view.findViewById(R.id.emailEditText);
76     streetEditText = (EditText) view.findViewById(R.id.streetEditText);
77     cityEditText = (EditText) view.findViewById(R.id.cityEditText);
78     stateEditText = (EditText) view.findViewById(R.id.stateEditText);
79     zipEditText = (EditText) view.findViewById(R.id.zipEditText);
80     contactInfoBundle = getArguments(); // null, se for a criação de um novo contato
81
82     if (contactInfoBundle != null)
83     {
84         rowID = contactInfoBundle.getLong(MainActivity.ROW_ID);
85         nameEditText.setText(contactInfoBundle.getString("name"));
86         phoneEditText.setText(contactInfoBundle.getString("phone"));
87         emailEditText.setText(contactInfoBundle.getString("email"));
88     }
89 }
90

```

Figura 8.38 Método sobrescrito onCreateView de AddEditFragment.

```

88         streetEditText.setText(contactInfoBundle.getString("street"));
89         cityEditText.setText(contactInfoBundle.getString("city"));
90         stateEditText.setText(contactInfoBundle.getString("state"));
91         zipEditText.setText(contactInfoBundle.getString("zip"));
92     }
93
94     // configura o receptor de eventos do componente Button Save Contact
95     Button saveContactButton =
96         (Button) view.findViewById(R.id.saveContactButton);
97     saveContactButton.setOnClickListener(saveContactButtonClicked);
98     return view;
99 }
100

```

Figura 8.38 Método sobrescrito onCreateView de AddEditFragment.

OnClickListener para processar eventos do componente Button Save Contact

Quando o usuário toca no botão **Save Contact**, o receptor de `saveContactButtonClicked` (Fig. 8.39) é executado. Para salvar um contato, o usuário precisa digitar pelo menos o nome do contato. O método `onClick` garante que o comprimento do nome seja maior que 0 caracteres (linha 107) e, se for, cria e executa uma `AsyncTask` para efetuar a operação de salvamento. O método `doInBackground` (linhas 113 a 118) chama `saveContact` (Fig. 8.40) para salvar o contato no banco de dados. O método `onPostExecute` (linhas 120 a 131) oculta o teclado via programação (linhas 124 a 128) e, então, notifica `MainActivity` de que um contato foi salvo (linha 130). Se `nameEditText` está vazio, as linhas 139 a 153 exibem um componente `DialogFragment` informando ao usuário que, para salvar o contato, um nome precisa ser fornecido.

```

101 // responde ao evento gerado quando o usuário salva um contato
102 OnClickListener saveContactButtonClicked = new OnClickListener()
103 {
104     @Override
105     public void onClick(View v)
106     {
107         if (nameEditText.getText().toString().trim().length() != 0)
108         {
109             // AsyncTask para salvar contato e, então, notificar o receptor
110             AsyncTask<Object, Object, Object> saveContactTask =
111                 new AsyncTask<Object, Object, Object>()
112                 {
113                     @Override
114                     protected Object doInBackground(Object... params)
115                     {
116                         saveContact(); // salva o contato no banco de dados
117                         return null;
118                     }
119
120                     @Override
121                     protected void onPostExecute(Object result)
122                     {
123                         // oculta o teclado virtual
124                         InputMethodManager imm = (InputMethodManager)
125                             getActivity().getSystemService(
126                                 Context.INPUT_METHOD_SERVICE);

```

Figura 8.39 OnClickListener para processar eventos do componente Button **Save Contact**. (continua)


```

127         imm.hideSoftInputFromWindow(
128             getView().getWindowToken(), 0);
129
130         listener.onAddEditCompleted(rowID);
131     }
132     }; // fim de AsyncTask
133
134     // salva o contato no banco de dados usando uma thread separada
135     saveContactTask.execute((Object[]) null);
136 }
137 else // o nome do contato obrigatório está em branco; portanto, exibe
138     // caixa de diálogo de erro
139 {
140     DialogFragment errorSaving =
141         new DialogFragment()
142     {
143         @Override
144         public Dialog onCreateDialog(Bundle savedInstanceState)
145         {
146             AlertDialog.Builder builder =
147                 new AlertDialog.Builder(getActivity());
148             builder.setMessage(R.string.error_message);
149             builder.setPositiveButton(R.string.ok, null);
150             return builder.create();
151         }
152     };
153     errorSaving.show(getFragmentManager(), "error saving contact");
154 }
155 } // fim do método onClick
156 }; // fim de OnClickListener saveContactButtonClicked
157

```

Figura 8.39 OnClickListener para processar eventos do componente Button Save Contact.

Método saveContact de AddEditFragment

O método saveContact (Fig. 8.40) salva as informações nos componentes EditText desse fragmento. Primeiramente, as linhas 162 e 163 criam o objeto DatabaseConnector e, então, verificam se contactInfoBundle é null. Se for, esse é um contato novo, e as linhas 168 a 175 obtêm os objetos String dos componentes EditText e os passam para o método insertContact do objeto DatabaseConnector para criar o novo contato. Se o objeto Bundle não é null, um contato já existente está sendo atualizado. Nesse caso, obtemos os objetos String dos componentes EditText e os passamos para o método updateContact do objeto DatabaseConnector usando o rowID existente para indicar o registro a ser atualizado. Os métodos insertContact e updateContact de DatabaseConnector tratam da abertura e do fechamento do banco de dados.

```

158     // salva informações de um contato no banco de dados
159     private void saveContact()
160     {
161         // obtém DatabaseConnector para interagir com o banco de dados SQLite
162         DatabaseConnector databaseConnector =
163             new DatabaseConnector(getActivity());
164
165         if (contactInfoBundle == null)
166         {
167             //insere as informações do contato no banco de dados

```

Figura 8.40 Método saveContact de AddEditFragment.

```

168         rowID = databaseConnector.insertContact(
169             nameEditText.getText().toString(),
170             phoneEditText.getText().toString(),
171             emailEditText.getText().toString(),
172             streetEditText.getText().toString(),
173             cityEditText.getText().toString(),
174             stateEditText.getText().toString(),
175             zipEditText.getText().toString());
176     }
177     else
178     {
179         databaseConnector.updateContact(rowID,
180             nameEditText.getText().toString(),
181             phoneEditText.getText().toString(),
182             emailEditText.getText().toString(),
183             streetEditText.getText().toString(),
184             cityEditText.getText().toString(),
185             stateEditText.getText().toString(),
186             zipEditText.getText().toString());
187     }
188 } // fim do método saveContact
189 } // fim da classe AddEditFragment

```

Figura 8.40 Método saveContact de AddEditFragment.

8.8 Classe DetailsFragment

A classe DetailsFragment (Figs. 8.41 a 8.50) exibe as informações de um contato e fornece itens de menu que permitem ao usuário editar ou excluir esse contato.

A instrução package e as instruções import de DetailsFragment

A Figura 8.41 lista a instrução package, as instruções import e o início da declaração da classe ContactListFragment. Nenhuma classe ou interface nova é usada nessa classe.

```

1 // DetailsFragment.java
2 // Exibe os detalhes de um contato
3 package com.deitel.addressbook;
4
5 import android.app.Activity;
6 import android.app.AlertDialog;
7 import android.app.Dialog;
8 import android.app.DialogFragment;
9 import android.app.Fragment;
10 import android.content.DialogInterface;
11 import android.database.Cursor;
12 import android.os.AsyncTask;
13 import android.os.Bundle;
14 import android.view.LayoutInflater;
15 import android.view.Menu;
16 import android.view.MenuInflater;
17 import android.view.MenuItem;
18 import android.view.View;
19 import android.view.ViewGroup;
20 import android.widget.TextView;
21
22 public class DetailsFragment extends Fragment
23 {

```

Figura 8.41 Instrução package e instruções import de DetailsFragment.

Interface DetailsFragmentManager

A Figura 8.42 declara a interface aninhada `DetailsFragmentManager` que contém os métodos de callback implementados por `MainActivity` para ser notificada quando o usuário excluir um contato (linha 28) e quando tocar no item de menu de edição para editar um contato (linha 31).

```
24 // métodos de callback implementados por MainActivity
25 public interface DetailsFragmentManager
26 {
27     // chamado quando um contato é excluído
28     public void onContactDeleted();
29
30     // chamado para passar objeto Bundle com informações de contato para edição
31     public void onEditContact(Bundle arguments);
32 }
33
```

Figura 8.42 Interface `DetailsFragmentManager`.

Variáveis de instância de DetailsFragment

A Figura 8.43 mostra as variáveis de instância da classe. A linha 34 declara a variável `listener`, a qual vai fazer referência ao objeto (`MainActivity`) que implementa a interface `DetailsFragmentManager`. A variável `rowID` representa o identificador de linha exclusivo do contato atual no banco de dados. As variáveis de instância de `TextView` (linhas 37 a 43) são usadas para exibir os dados do contato na tela.

```
34 private DetailsFragmentManager listener;
35
36 private long rowID = -1; // rowID do contato selecionado
37 private TextView nameTextView; // exibe o nome do contato
38 private TextView phoneTextView; // exibe o telefone do contato
39 private TextView emailTextView; // exibe o e-mail do contato
40 private TextView streetTextView; // exibe a rua do contato
41 private TextView cityTextView; // exibe a cidade do contato
42 private TextView stateTextView; // exibe o estado do contato
43 private TextView zipTextView; // exibe o código postal do contato
44
```

Figura 8.43 Variáveis de instância de `DetailsFragment`.

Métodos sobrescritos onAttach e onDetach de DetailsFragment

A classe `DetailsFragment` sobrepõe os métodos de ciclo de vida `onAttach` e `onDetach` de `Fragment` (Fig. 8.44) para configurar a variável de instância `listener` quando `DetailsFragment` é anexado e desanexado, respectivamente.

```
45 // configura DetailsFragmentManager quando o fragmento é anexado
46 @Override
47 public void onAttach(Activity activity)
48 {
49     super.onAttach(activity);
50     listener = (DetailsFragmentManager) activity;
51 }
52
```

Figura 8.44 Métodos sobrescritos `onAttach` e `onDetach` de `DetailsFragment`.

```

53 // remove DetailsFragmentListener quando o fragmento é desanexado
54 @Override
55 public void onDetach()
56 {
57     super.onDetach();
58     listener = null;
59 }
60

```

Figura 8.44 Métodos sobrescritos onAttach e onDetach de DetailsFragment.

Método sobrescrito onCreateView de DetailsFragment

O método onCreateView (Fig. 8.45) obtém o identificador de linha do contato selecionado (linhas 70 a 79). Se o fragmento está sendo restaurado, carregamos o rowID do bundle savedInstanceState; caso contrário, o obtemos do Bundle de argumentos de Fragment. As linhas 82 a 93 inflam a interface gráfica do usuário e obtêm referências para os componentes TextView.

```

61 // chamado quando a view de DetailsFragmentListener precisa ser criada
62 @Override
63 public View onCreateView(LayoutInflater inflater, ViewGroup container,
64     Bundle savedInstanceState)
65 {
66     super.onCreateView(inflater, container, savedInstanceState);
67     setRetainInstance(true); // salva o fragmento entre mudanças de configuração
68
69     // se DetailsFragment está sendo restaurado, obtém o identificador de linha salvo
70     if (savedInstanceState != null)
71         rowID = savedInstanceState.getLong(MainActivity.ROW_ID);
72     else
73     {
74         // obtém o Bundle de argumentos e extrai o identificador de linha do contato
75         Bundle arguments = getArguments();
76
77         if (arguments != null)
78             rowID = arguments.getLong(MainActivity.ROW_ID);
79     }
80
81     // infla o layout de DetailsFragment
82     View view =
83         inflater.inflate(R.layout.fragment_details, container, false);
84     setHasOptionsMenu(true); // este fragmento tem itens de menu a exibir
85
86     // obtém os componentes EditText
87     nameTextView = (TextView) view.findViewById(R.id.nameTextView);
88     phoneTextView = (TextView) view.findViewById(R.id.phoneTextView);
89     emailTextView = (TextView) view.findViewById(R.id.emailTextView);
90     streetTextView = (TextView) view.findViewById(R.id.streetTextView);
91     cityTextView = (TextView) view.findViewById(R.id.cityTextView);
92     stateTextView = (TextView) view.findViewById(R.id.stateTextView);
93     zipTextView = (TextView) view.findViewById(R.id.zipTextView);
94     return view;
95 }
96

```

Figura 8.45 Método sobrescrito onCreateView de DetailsFragment.

Método sobrescrito *onResume* de *DetailsFragment*

O método de ciclo de vida *onResume* de *Fragment* (Fig. 8.46) cria e executa um objeto *AsyncTask* (linha 102), do tipo *LoadContactTask* (definido na Fig. 8.49), que obtém o contato especificado do banco de dados e exibe seus dados. Neste caso, o argumento do método *execute* é o *rowID* do contato a ser carregado. Sempre que a linha 102 é executada, ela cria um novo objeto *LoadContactTask* – novamente, isso é obrigatório, pois cada *AsyncTask* pode ser executada *apenas uma vez*.

```

97      // chamado quando o elemento DetailsFragment recomeça
98      @Override
99      public void onResume()
100     {
101         super.onResume();
102         new LoadContactTask().execute(rowID); // carrega o contato em rowID
103     }
104 
```

Figura 8.46 Método sobrescrito *onResume* de *DetailsFragment*.

Método sobrescrito *onSaveInstanceState* de *DetailsFragment*

O método *onSaveInstanceState* de *Fragment* (Fig. 8.47) salva o *rowID* do contato selecionado quando a configuração do dispositivo muda durante a execução do aplicativo – por exemplo, quando o usuário gira o dispositivo ou abre o teclado em um dispositivo com teclado físico. O estado dos componentes da interface gráfica do usuário é salvo automaticamente, mas quaisquer outros itens que você queira restaurar durante uma mudança de configuração devem ser armazenados no objeto *Bundle* recebido por *onSaveInstanceState*.

```

105     // salva o identificador de linha do contato que está sendo exibido
106     @Override
107     public void onSaveInstanceState(Bundle outState)
108     {
109         super.onSaveInstanceState(outState);
110         outState.putLong(MainActivity.ROW_ID, rowID);
111     }
112 
```

Figura 8.47 Método sobrescrito *onSaveInstanceState* de *DetailsFragment*.

Métodos sobrescritos *onOptionsItemSelected* e *onOptionsItemSelected* de *DetailsFragment*

O menu de *DetailsFragment* fornece opções para editar o contato atual e para excluí-lo. O método *onOptionsItemSelected* (Fig. 8.48, linhas 114 a 119) infla o arquivo de recursos de menu *fragment_details_menu.xml*. O método *onOptionsItemSelected* (linhas 122 a 146) usa o identificador de recurso do componente *MenuItem* selecionado para determinar qual deles foi selecionado. Se o usuário selecionou o item de menu com identificador *R.id.action_edit*, as linhas 129 a 137 criam um objeto *Bundle* contendo os dados do contato e, então, a linha 138 passa esse objeto para *DetailsFragmentListener* a fim de ser usado no componente *AddEditFragment*. Se o usuário selecionou o item de menu com identificador *R.id.action_delete*, a linha 141 chama o método *deleteContact* (Fig. 8.50).

```

113 // exibe os seleções de menu deste fragmento
114 @Override
115 public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
116 {
117     super.onCreateOptionsMenu(menu, inflater);
118     inflater.inflate(R.menu.fragment_details_menu, menu);
119 }
120
121 // trata as seleções de item de menu
122 @Override
123 public boolean onOptionsItemSelected(MenuItem item)
124 {
125     switch (item.getItemId())
126     {
127         case R.id.action_edit:
128             // cria objeto Bundle contendo os dados do contato a editar
129             Bundle arguments = new Bundle();
130             arguments.putLong(MainActivity.ROW_ID, rowID);
131             arguments.putCharSequence("name", nameTextView.getText());
132             arguments.putCharSequence("phone", phoneTextView.getText());
133             arguments.putCharSequence("email", emailTextView.getText());
134             arguments.putCharSequence("street", streetTextView.getText());
135             arguments.putCharSequence("city", cityTextView.getText());
136             arguments.putCharSequence("state", stateTextView.getText());
137             arguments.putCharSequence("zip", zipTextView.getText());
138             listener.onEditContact(arguments); // passa objeto Bundle para o receptor
139             return true;
140         case R.id.action_delete:
141             deleteContact();
142             return true;
143     }
144
145     return super.onOptionsItemSelected(item);
146 }
147

```

Figura 8.48 Métodos sobrescritos `onCreateOptionsMenu` e `onOptionsItemSelected` de `DetailsFragment`.

Subclasse `LoadContactTask` de `AsyncTask`

A classe aninhada `LoadContactTask` (Fig. 8.49) estende a classe `AsyncTask` e define como interagir com o banco de dados para obter as informações de um contato para exibição. Neste caso, os três parâmetros de tipo genérico são:

- Long para a lista de argumentos de comprimento variável passada para o método `doInBackground` de `AsyncTask`. Isso vai conter o identificador de linha necessário para localizar um contato.
- Object para a lista de argumentos de comprimento variável passada para o método `onProgressUpdate` de `AsyncTask`, o qual não usamos neste exemplo.
- Cursor para o tipo do resultado da tarefa, que é passado para o método `onPostExecute` de `AsyncTask`.

```

148 // executa a consulta de banco de dados fora da thread da interface gráfica
149 // do usuário
149 private class LoadContactTask extends AsyncTask<Long, Object, Cursor>
150 {
151     DatabaseConnector databaseConnector =
152         new DatabaseConnector(getActivity());

```

Figura 8.49 Subclasse `LoadContactTask` de `AsyncTask`. (continua)

```

153
154 // abre o banco de dados e obtém o objeto Cursor que representa os dados do
// contato especificado
155 @Override
156 protected Cursor doInBackground(Long... params)
157 {
158     databaseConnector.open();
159     return databaseConnector.getOneContact(params[0]);
160 }
161
162 // usa o Cursor retornado pelo método doInBackground
163 @Override
164 protected void onPostExecute(Cursor result)
165 {
166     super.onPostExecute(result);
167     result.moveToFirst(); // move para o primeiro item
168
169     // obtém o índice de coluna de cada item de dado
170     int nameIndex = result.getColumnIndex("name");
171     int phoneIndex = result.getColumnIndex("phone");
172     int emailIndex = result.getColumnIndex("email");
173     int streetIndex = result.getColumnIndex("street");
174     int cityIndex = result.getColumnIndex("city");
175     int stateIndex = result.getColumnIndex("state");
176     int zipIndex = result.getColumnIndex("zip");
177
178     // preenche os componentes TextView com os dados recuperados
179     nameTextView.setText(result.getString(nameIndex));
180     phoneTextView.setText(result.getString(phoneIndex));
181     emailTextView.setText(result.getString(emailIndex));
182     streetTextView.setText(result.getString(streetIndex));
183     cityTextView.setText(result.getString(cityIndex));
184     stateTextView.setText(result.getString(stateIndex));
185     zipTextView.setText(result.getString(zipIndex));
186
187     result.close(); // fecha o cursor de resultado
188     databaseConnector.close(); // fecha a conexão de banco de dados
189 } // fim do método onPostExecute
190 } // fim da classe LoadContactTask
191

```

Figura 8.49 Subclasse LoadContactTask de AsyncTask.

As linhas 151 e 152 criam um novo objeto de nossa classe DatabaseConnector (Seção 8.9). O método `doInBackground` (linhas 155 a 160) abre a conexão com o banco de dados e chama o método `getOneContact` de DatabaseConnector, o qual consulta o banco de dados para obter o contato com o `rowID` especificado, que foi passado como único argumento para o método `execute` de AsyncTask. Em `doInBackground`, `rowID` é armazenado em `params[0]`.

O objeto Cursor resultante é passado para o método `onPostExecute` (linhas 163 a 189). O cursor é posicionado *antes* da primeira linha do conjunto resultante (`result set`). Nesse caso, o conjunto vai conter apenas um registro, de modo que o método `moveToFirst` de Cursor (linha 167) pode ser usado para mover o cursor para a primeira linha no conjunto resultante. [*Obs.:* é considerado uma boa prática garantir que o método `moveToFirst` de Cursor retorne `true` antes de tentar obter dados do Cursor. Neste aplicativo, sempre vai haver uma linha no Cursor.]

Usamos o método `getColumnIndex` de Cursor (linhas 170 a 176) para obter os índices das colunas na tabela `contacts` do banco de dados. (Codificamos os nomes de coluna neste aplicativo, mas eles poderiam ser implementados como constantes String, como fizemos para `ROW_ID` na classe MainActivity, na Fig. 8.14.) Esse método retorna -1 se a

coluna não estiver no resultado da consulta. A classe `Cursor` também fornece o método `getColumnIndexOrThrow`, caso você prefira obter uma exceção quando o nome de coluna especificado não existir. As linhas 179 a 185 usam o método `getString` de `Cursor` para recuperar os valores `String` das colunas do `cursor` e, então, exibem esses valores nos componentes `TextView` correspondentes. As linhas 187 e 188 fecham o `Cursor` e a conexão com o banco de dados, pois eles não são mais necessários. É considerado uma boa prática liberar recursos, como conexões de banco de dados, quando não estão sendo usados, para que outras atividades possam utilizá-los.

Método `deleteContact` e `confirmDelete` de `DialogFragment`

O método `deleteContact` (Fig. 8.50, linhas 193 a 197) exibe um elemento `DialogFragment` (linhas 200 a 252) solicitando ao usuário que confirme se o contato exibido no momento deve ser excluído. Em caso positivo, `DialogFragment` usa um objeto `AsyncTask` para excluir o contato do banco de dados. Se o usuário clicar no botão **Delete** na caixa de diálogo, as linhas 222 e 223 criam um novo objeto `DatabaseConnector`. As linhas 226 a 241 criam uma `AsyncTask` que, quando executada (linha 244), passa um valor `Long` representando o identificador de linha do contato para `doInBackground`, o qual então exclui o contato. A linha 232 chama o método `deleteContact` de `DatabaseConnector` para fazer a exclusão. Quando `doInBackground` termina de ser executado, a linha 239 chama o método `onContactDeleted` de `listener` para que `MainActivity` possa remover o componente `DetailsFragment` da tela.

```

192 // exclui um contato
193 private void deleteContact()
194 {
195     // usa FragmentManager para exibir o componente DialogFragment de confirmDelete
196     confirmDelete.show(getFragmentManager(), "confirm delete");
197 }
198
199 // DialogFragment para confirmar a exclusão de contato
200 private DialogFragment confirmDelete =
201     new DialogFragment()
202     {
203         // cria um componente AlertDialog e o retorna
204         @Override
205         public Dialog onCreateDialog(Bundle bundle)
206         {
207             // cria um novo AlertDialog Builder
208             AlertDialog.Builder builder =
209                 new AlertDialog.Builder(getActivity());
210
211             builder.setTitle(R.string.confirm_title);
212             builder.setMessage(R.string.confirm_message);
213
214             // fornece um botão OK que simplesmente descarta a caixa de diálogo
215             builder.setPositiveButton(R.string.button_delete,
216                 new DialogInterface.OnClickListener()
217                 {
218                     @Override
219                     public void onClick(
220                         DialogInterface dialog, int button)
221                     {
222                         final DatabaseConnector databaseConnector =
223                             new DatabaseConnector(getActivity());
224

```

Figura 8.50 Método `deleteContact` e `confirmDelete` de `DialogFragment`. (continua)


```

225         // AsyncTask exclui contato e notifica o receptor
226         AsyncTask<Long, Object, Object> deleteTask =
227             new AsyncTask<Long, Object, Object>()
228             {
229                 @Override
230                 protected Object doInBackground(Long... params)
231                 {
232                     databaseConnector.deleteContact(params[0]);
233                     return null;
234                 }
235
236                 @Override
237                 protected void onPostExecute(Object result)
238                 {
239                     listener.onContactDeleted();
240                 }
241             }; // fim de AsyncTask
242
243         // executa AsyncTask para excluir o contato em rowID
244         deleteTask.execute(new Long[] { rowID });
245     } // fim do método onClick
246 } // fim da classe interna anônima
247 ); // fim da chamada ao método setPositiveButton
248
249 builder.setNegativeButton(R.string.button_cancel, null);
250 return builder.create(); // retorna o componente AlertDialog
251 }
252 }; // fim da classe interna anônima de DialogFragment
253 } // fim da classe DetailsFragment

```

Figura 8.50 Método deleteContact e confirmDelete de DialogFragment.

8.9 Classe utilitária DatabaseConnector

A classe utilitária DatabaseConnector (Figs. 8.51 a 8.58) gerencia as interações deste aplicativo com o SQLite para criar e manipular o banco de dados UserContacts, o qual contém uma tabela chamada contacts.

Instrução package, instruções import e campos

A Figura 8.51 lista a instrução package, as instruções import e os campos da classe DatabaseConnector. Realçamos as instruções import das novas classes e interfaces discutidas na Seção 8.3. A constante String DATABASE_NAME (linha 16) especifica o nome do banco de dados que vai ser criado ou aberto. *Os nomes de banco de dados devem ser exclusivos dentro de um aplicativo específico, mas de um aplicativo para outro não precisam ser.* Um objeto SQLiteDatabase (linha 18) fornece acesso de leitura/gravação para um banco de dados SQLite. DatabaseOpenHelper (linha 19) é uma classe privada aninhada que estende a classe abstrata SQLiteOpenHelper – essa classe é usada para gerenciar a criação, a abertura e a atualização (upgrade) de bancos de dados (talvez para modificar a estrutura de um banco de dados). Discutimos SQLiteOpenHelper com mais detalhes na Fig. 8.58.

```

1 // DatabaseConnector.java
2 // Fornece fácil conexão e criação do banco de dados UserContacts.
3 package com.deitel.addressbook;
4
5 import android.content.ContentValues;

```

Figura 8.51 Instrução package, instruções import e variáveis de instância da classe DatabaseConnector.

```

6 import android.content.Context;
7 import android.database.Cursor;
8 import android.database.SQLException;
9 import android.database.sqlite.SQLiteDatabase;
10 import android.database.sqlite.SQLiteOpenHelper;
11 import android.database.sqlite.SQLiteDatabase.CursorFactory;
12
13 public class DatabaseConnector
14 {
15     // nome do banco de dados
16     private static final String DATABASE_NAME = "UserContacts";
17
18     private SQLiteDatabase database; // para interagir com o banco de dados
19     private DatabaseOpenHelper databaseOpenHelper; // cria o banco de dados
20

```

Figura 8.51 Instrução package, instruções import e variáveis de instância da classe DatabaseConnector.

Construtor e métodos open e close de DatabaseConnector

O construtor de DatabaseConnector (Fig. 8.52, linhas 22 a 27) cria um novo objeto da classe DatabaseOpenHelper (Fig. 8.58), o qual vai ser usado para abrir ou criar o banco de dados. Discutimos os detalhes do construtor DatabaseOpenHelper na Figura 8.58. O método open (linhas 30 a 34) tenta estabelecer uma conexão com o banco de dados e lança uma exceção SQLException caso a tentativa de conexão falhe. O método **getWritableDatabase** (linha 33), herdado de SQLiteOpenHelper, retorna um objeto SQLiteDatabase. Se o banco de dados ainda não foi criado, esse método o cria; caso contrário, o abre. Uma vez aberto o banco de dados, ele é *colocado na cache* pelo sistema operacional para melhorar o desempenho de futuras interações com o banco de dados. O método close (linhas 37 a 41) fecha a conexão com o banco de dados, chamando o método herdado close de SQLiteOpenHelper.

```

21 // construtor public de DatabaseConnector
22 public DatabaseConnector(Context context)
23 {
24     // cria um novo DatabaseOpenHelper
25     databaseOpenHelper =
26         new DatabaseOpenHelper(context, DATABASE_NAME, null, 1);
27 }
28
29 // abre a conexão de banco de dados
30 public void open() throws SQLException
31 {
32     // cria ou abre um banco de dados para leitura/gravação
33     database = databaseOpenHelper.getWritableDatabase();
34 }
35
36 // fecha a conexão do banco de dados
37 public void close()
38 {
39     if (database != null)
40         database.close(); // fecha a conexão do banco de dados
41 }
42

```

Figura 8.52 Construtor e métodos open e close de DatabaseConnector.

Método *insertContact* de *DatabaseConnector*

O método `insertContact` (Fig. 8.53) insere no banco de dados um novo contato com as informações dadas. Primeiramente, colocamos as informações do contato em um novo objeto `ContentValues` (linhas 47 a 54), o qual mantém um mapa de pares chave-valor – os nomes das colunas do banco de dados são as chaves. As linhas 56 a 58 abrem o banco de dados, inserem o novo contato e fecham o banco de dados. O **método** `insert` de `SQLiteDatabase` (linha 57) insere os valores de `ContentValues` fornecidos na tabela especificada como primeiro argumento – a tabela "contacts", neste caso. O segundo parâmetro desse método, que não é utilizado neste aplicativo, é chamado `nullColumnHack` e não é necessário porque o *SQLite não aceita a inserção de uma linha completamente vazia na tabela* – isso seria equivalente a passar um objeto `ContentValues` vazio para `insert`. Em vez de tornar inválido passar `ContentValues` vazio para o método, o parâmetro `nullColumnHack` é usado para identificar uma coluna que aceite valores `NULL`.

```

43 // insere um novo contato no banco de dados
44 public long insertContact(String name, String phone, String email,
45 String street, String city, String state, String zip)
46 {
47     ContentValues newContact = new ContentValues();
48     newContact.put("name", name);
49     newContact.put("phone", phone);
50     newContact.put("email", email);
51     newContact.put("street", street);
52     newContact.put("city", city);
53     newContact.put("state", state);
54     newContact.put("zip", zip);
55
56     open(); // abre o banco de dados
57     long rowID = database.insert("contacts", null, newContact);
58     close(); // fecha o banco de dados
59     return rowID;
60 } // fim do método insertContact
61

```

Figura 8.53 Método `insertContact` de `DatabaseConnector`.

Método *updateContact* de *DatabaseConnector*

O método `updateContact` (Fig. 8.54) é semelhante ao método `insertContact`, a não ser pelo fato de chamar o **método** `update` de `SQLiteDatabase` (linha 76) para atualizar um contato já existente. O terceiro argumento do método `update` representa uma cláusula `WHERE` em SQL (sem a palavra-chave `WHERE`) que especifica o(s) registro(s) a atualizar. Neste caso, usamos o identificador de linha do registro para atualizar um contato específico.

```

62 // atualiza um contato existente no banco de dados
63 public void updateContact(long id, String name, String phone,
64 String email, String street, String city, String state, String zip)
65 {
66     ContentValues editContact = new ContentValues();
67     editContact.put("name", name);

```

Figura 8.54 Método `updateContact` de `DatabaseConnector`.

```

68     editContact.put("phone", phone);
69     editContact.put("email", email);
70     editContact.put("street", street);
71     editContact.put("city", city);
72     editContact.put("state", state);
73     editContact.put("zip", zip);
74
75     open(); // abre o banco de dados
76     database.update("contacts", editContact, "_id=" + id, null);
77     close(); // fecha o banco de dados
78 }
79

```

Figura 8.54 Método updateContact de DatabaseConnector.

Método getAllContacts

O método getAllContacts (Fig. 8.55) usa o **método query** de SQLiteDatabase (linhas 83 e 84) para recuperar um Cursor que dá acesso aos identificadores e nomes de todos os contatos do banco de dados. Os argumentos são:

- O nome da tabela a ser consultada.
- Um array de Strings dos nomes de coluna a retornar (as colunas _id e name, aqui) – null retorna todas as colunas da tabela, o que geralmente é considerado uma prática de programação ruim, pois, para economizar memória, tempo de processador e energia da bateria, você deve obter apenas os dados necessários.
- Uma cláusula WHERE em SQL (sem a palavra-chave WHERE), ou null para retornar todas as linhas.
- Um array de Strings de argumentos a serem substituídos na cláusula WHERE, sempre que ? for usado como espaço reservado para o valor de um argumento, ou null, caso não existam argumentos na cláusula WHERE.
- Uma cláusula GROUP BY em SQL (sem as palavras-chave GROUP BY), ou null, se você não quiser agrupar os resultados.
- Uma cláusula HAVING em SQL (sem a palavra-chave HAVING) para especificar quais grupos da cláusula GROUP BY vão ser incluídos nos resultados – null é exigido se a cláusula GROUP BY for null.
- Uma cláusula ORDER BY em SQL (sem as palavras-chave ORDER BY) para especificar a ordem dos resultados, ou null, caso você não queira especificar a ordem.

O objeto Cursor retornado pelo método query contém todas as linhas da tabela que correspondem aos argumentos do método – o assim chamado *conjunto resultante*. O cursor é posicionado *antes* da primeira linha do conjunto resultante – os vários métodos move do Cursor podem ser usados para mover o cursor pelo conjunto resultante para processamento.

```

80 // retorna um Cursor com todos os nomes de contato do banco de dados
81 public Cursor getAllContacts()
82 {
83     return database.query("contacts", new String[] { "_id",
84         null, null, null, null, "name" });
85 }
86

```

Figura 8.55 Método getAllContacts de DatabaseConnector.

Método *getOneContact*

O método `getOneContact` (Fig. 8.56) também usa o método `query` de `SqliteDatabase` para consultar o banco de dados. Neste caso, recuperamos todas as colunas do banco de dados para o contato com o identificador especificado.

```
87 // retorna um Cursor contendo as informações do contato especificado
88 public Cursor getOneContact(long id)
89 {
90     return database.query(
91         "contacts", null, "_id=" + id, null, null, null, null);
92 }
93
```

Figura 8.56 Método `getOneContact` de `DatabaseConnector`.

Método *deleteContact*

O método `deleteContact` (Fig. 8.57) usa o método `delete` de `SqliteDatabase` (linha 98) para excluir um contato do banco de dados. Neste caso, recuperamos todas as colunas do banco de dados para o contato com o identificador especificado. Os três argumentos são: a tabela do banco de dados da qual o registro vai ser excluído, a cláusula `WHERE` (sem a palavra-chave `WHERE`) e, se essa cláusula tiver argumentos, um array de `Strings` dos valores a serem substituídos na cláusula `WHERE` (`null`, em nosso caso).

```
94 // exclui o contato especificado pelo nome String fornecido
95 public void deleteContact(long id)
96 {
97     open(); // abre o banco de dados
98     database.delete("contacts", "_id=" + id, null);
99     close(); // fecha o banco de dados
100 }
101
```

Figura 8.57 Método `deleteContact` de `DatabaseConnector`.

Classe aninhada privada *DatabaseOpenHelper* que estende *SQLiteOpenHelper*

A classe privada `DatabaseOpenHelper` (Fig. 8.58) estende a classe abstrata `SQLiteOpenHelper`, a qual ajuda os aplicativos a criar bancos de dados e a gerenciar mudanças de versão. O construtor (linhas 105 a 109) simplesmente chama o construtor da superclasse, o qual exige quatro argumentos:

- o objeto `Context` no qual o banco de dados está sendo criado ou aberto;
- o nome do banco de dados – pode ser `null`, caso você queira usar um banco de dados na memória;
- o objeto `CursorFactory` a ser usado – `null` indica que você quer usar `CursorFactory` padrão do `SQLite` (normalmente usado para a maioria dos aplicativos); e
- o número de versão do banco de dados (começando com 1).

Você deve sobrescrever os métodos abstratos `onCreate` e `onUpgrade` dessa classe. Se o banco de dados ainda não existe, o método `onCreate` de `DatabaseOpenHelper` vai ser chamado para criá-lo. Se você fornecer um número de versão mais recente

do que a versão do banco de dados que está armazenada no dispositivo, o **método onUpgrade** de `DatabaseOpenHelper` vai ser chamado para migrar o banco de dados para a nova versão (talvez para adicionar tabelas ou para adicionar colunas em uma tabela já existente).

```

102 private class DatabaseOpenHelper extends SQLiteOpenHelper
103 {
104     // construtor
105     public DatabaseOpenHelper(Context context, String name,
106         CursorFactory factory, int version)
107     {
108         super(context, name, factory, version);
109     }
110
111     // cria a tabela de contatos quando o banco de dados é gerado
112     @Override
113     public void onCreate(SQLiteDatabase db)
114     {
115         // consulta para criar uma nova tabela chamada contacts
116         String createQuery = "CREATE TABLE contacts" +
117             "(_id integer primary key autoincrement," +
118             "name TEXT, phone TEXT, email TEXT, " +
119             "street TEXT, city TEXT, state TEXT, zip TEXT);";
120
121         db.execSQL(createQuery); // executa a consulta para criar o banco de dados
122     }
123
124     @Override
125     public void onUpgrade(SQLiteDatabase db, int oldVersion,
126         int newVersion)
127     {
128     }
129 } // fim da classe DatabaseOpenHelper
130 } // fim da classe DatabaseConnector

```

Figura 8.58 Classe `DatabaseOpenHelper` de `SQLiteOpenHelper`.

O método `onCreate` (linhas 112 a 122) especifica a tabela a ser criada com o comando SQL `CREATE TABLE`, o qual é definido como uma `String` (linhas 116 a 119). Neste caso, a tabela de contatos contém um campo de chave primária inteiro (`_id`) que é incrementado automaticamente e os campos de texto de todas as outras colunas. A linha 121 usa o método `execSQL` de `SQLiteDatabase` para executar o comando `CREATE TABLE`. Como não precisamos migrar o banco de dados, simplesmente sobrescrevemos o método `onUpgrade` com um corpo vazio. A classe `SQLiteOpenHelper` também fornece o **método onDowngrade**, que pode ser usado para rebaixar um banco de dados quando a versão correntemente armazenada tem um número mais alto do que o solicitado na chamada do construtor da classe `SQLiteOpenHelper`. O rebaixamento pode ser usado para reverter o banco de dados para uma versão anterior, com menos colunas em uma tabela ou menos tabelas no banco de dados – talvez para corrigir um erro no aplicativo.

Todos os métodos de `SQLiteDatabase` que utilizamos na classe `DatabaseConnector` têm métodos correspondentes que efetuam as mesmas operações, mas que lançam exceções em caso de falha, em vez de apenas retornar `-1` (por exemplo, `insertOrThrow` *versus* `insert`). Esses métodos são intercambiáveis, permitindo que você decida como lidar com erros de leitura e gravação no banco de dados.

8.10 Para finalizar

Neste capítulo, você criou o aplicativo **Address Book**, o qual permite aos usuários adicionar, ver, editar e excluir informações de contato armazenadas em um banco de dados SQLite. Você definiu pares atributo-valor comuns de componentes de interface gráfica do usuário, como recursos `style` em XML e, então, aplicou os estilos em todos os componentes que compartilham esses valores, usando o atributo `style` dos componentes. Você adicionou uma borda a um componente `TextView` especificando `Drawable` como valor do atributo `android:background` desse componente e criou um objeto `Drawable` personalizado usando uma representação XML de um elemento `shape`. Usou também ícones padrão do Android para melhorar a aparência dos itens do menu do aplicativo.

Quando a principal tarefa de um fragmento é exibir uma lista de itens que pode ser rolada, você aprendeu que pode estender a classe `ListFragment` para criar um fragmento que exibe um componente `ListView` em seu layout padrão. Isso foi feito para exibir os contatos armazenados no banco de dados do aplicativo. Você vinculou dados ao componente `ListView` por meio de um elemento `CursorAdapter` que exibia os resultados de uma consulta de banco de dados.

Na atividade deste aplicativo, você usou `FragmentTransactions` para adicionar e substituir fragmentos dinamicamente na interface gráfica do usuário. Usou também a pilha de retrocesso de fragmentos para dar suporte ao botão *voltar*, a fim de voltar para um fragmento exibido anteriormente e permitir que a atividade do aplicativo retornasse a fragmentos anteriores via programação.

Demonstramos como comunicar dados entre fragmentos e uma atividade hospedeira ou outros fragmentos da atividade por meio de interfaces de métodos de callback implementados pela atividade hospedeira. Você também usou objetos `Bundle` para passar argumentos para os fragmentos.

Você usou uma subclasse de `SQLiteOpenHelper` a fim de simplificar a criação do banco de dados e a fim de obter um objeto `SQLiteDatabase` para manipular o conteúdo de um banco de dados. Você processou resultados de consulta por meio de um `Cursor` e usou subclasses de `AsyncTask` para executar tarefas de banco de dados fora da thread da interface gráfica do usuário e retornar os resultados para essa thread. Isso permitiu tirar proveito dos recursos de thread do Android sem criar e manipular threads diretamente.

No Capítulo 9, vamos discutir o lado comercial do desenvolvimento de aplicativos Android. Você também vai ver como preparar seu aplicativo para enviar ao Google Play, incluindo a produção de ícones. Vamos discutir como testar seus aplicativos em dispositivos e publicá-los no Google Play. Discutiremos as características de aplicativos notáveis e as diretrizes de projeto do Android a serem seguidas. Forneceremos dicas para fixação de preço e comercialização de seu aplicativo. Examinaremos também as vantagens de oferecer seu aplicativo gratuitamente para alavancar as vendas de outros produtos, como uma versão mais completa do aplicativo ou conteúdo melhor. Mostraremos como usar o Google Play para monitorar as vendas do aplicativo, os pagamentos e muito mais.

Exercícios de revisão

8.1 Preencha os espaços em branco em cada um dos seguintes enunciados:

- a) Os resultados da consulta ao banco de dados SQLite são gerenciados por meio de um _____ (pacote `android.database`).

- b) Para obter os resultados de uma operação de banco de dados na thread da interface gráfica, você usa uma _____ (pacote `android.os`) para efetuar a operação em uma thread e receber os resultados na thread da interface.
 - c) O método _____ de `Fragment` retorna o objeto `Bundle` de argumentos para o fragmento.
 - d) O objeto `Cursor` retornado pelo método `query` contém todas as linhas da tabela que correspondem aos argumentos do método – o assim chamado _____.
- 8.2** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- a) Considera-se uma boa prática liberar recursos, como conexões de banco de dados, quando não estão sendo usados, para que outras atividades possam utilizá-los.
 - b) Considera-se uma boa prática garantir que o método `moveToFirst` de `Cursor` retorne `false` antes de tentar obter dados do objeto `Cursor`.
 - c) Considera-se uma boa prática efetuar operações longas ou operações que bloqueiam a execução até que terminem (por exemplo, acesso a arquivos e a bancos de dados) na thread da interface gráfica do usuário.
 - d) `SimpleCursorAdapter` é uma subclasse de `CursorAdapter` projetada para simplificar o mapeamento de colunas de um `Cursor` diretamente nos componentes `TextView` ou `ImageView` definidos em seus layouts XML.
 - e) Uma vantagem importante de usar `SyncTask` é que esse elemento trata dos detalhes da criação de threads e da execução de seus métodos nas threads apropriadas, de modo que você não precisa interagir com o mecanismo de threads diretamente.

Respostas dos exercícios de revisão

- 8.1** a) `Cursor`. b) `AsyncTask`. c) `getArguments`. d) conjunto resultante.
- 8.2** a) Verdadeira. b) Falsa. Considera-se uma boa prática garantir que o método `moveToFirst` de `Cursor` retorne `true` antes de tentar obter dados do objeto `Cursor`. c) Falsa. Considera-se uma boa prática efetuar operações longas ou operações que bloqueiam a execução até que terminem (por exemplo, acesso a arquivos e a bancos de dados) *fora* da thread da interface gráfica do usuário. d) Verdadeira. e) Falsa. Uma vantagem importante de usar `AsyncTask` é que esse elemento trata dos detalhes da criação de threads e da execução de seus métodos nas threads apropriadas, de modo que você não precisa interagir com o mecanismo de threads diretamente.

Exercícios

- 8.3** (*Modificação do aplicativo Flag Quiz*) Revise o aplicativo `Flag Quiz` para usar apenas uma atividade, fragmentos dinâmicos e objetos `FragmentTransaction`, como você fez no aplicativo `Address Book`.
- 8.4** (*Aplicativo Movie Collection*) Usando as técnicas aprendidas neste capítulo, crie um aplicativo que permita digitar informações sobre sua coleção de filmes. Forneça campos para o título, ano, diretor e quaisquer outros campos que você queira registrar. O aplicativo deve fornecer atividades semelhantes ao aplicativo `Address Book` para ver a lista de filmes (em ordem alfabética), adicionar e/ou atualizar as informações de um filme e ver os seus detalhes.
- 8.5** (*Aplicativo Recipe*) Usando as técnicas aprendidas neste capítulo, crie um aplicativo de receitas culinárias. Forneça campos para o nome da receita, categoria (por exemplo, aperitivo, entrada, sobremesa, salada, acompanhamento), a lista dos ingredientes e instruções de preparo. O aplicativo deve fornecer atividades semelhantes ao aplicativo `Address Book` para

ver a lista de receitas (em ordem alfabética), adicionar e/ou atualizar as informações de uma receita e ver os detalhes de uma receita.

- 8.6 (*Aprimoramento do aplicativo Twitter Searches*)** Usando as técnicas aprendidas neste capítulo, modifique o aplicativo *Favorite Twitter Searches* de modo que ele carregue e salve os componentes *SharedPreferences* em uma thread de execução separada.
- 8.7 (*Aplicativo Shopping List*)** Crie um aplicativo que permita ao usuário digitar e editar uma lista de compras. Inclua um recurso de favoritos que permita ao usuário adicionar facilmente os itens comprados com frequência. Inclua um recurso opcional para inserir o preço e a quantidade de cada item, para que o usuário possa monitorar o custo total de todos os itens da lista.
- 8.8 (*Aplicativo Expense Tracker*)** Crie um aplicativo que permita ao usuário controlar suas despesas pessoais. Forneça categorias para classificar cada despesa (por exemplo, despesas mensais, viagens, entretenimento, necessidades). Forneça uma opção para identificar despesas recorrentes e as inclua automaticamente em um calendário na frequência apropriada (diária, semanal, mensal ou anual). Opcional: investigue o mecanismo de notificações na barra de status do Android em developer.android.com/guide/topics/ui/notifiers/index.html. Forneça notificações para lembrar o usuário sobre o vencimento de uma fatura.
- 8.9 (*Aplicativo Cooking with Healthier Ingredients*)** Nos Estados Unidos, a obesidade está aumentando em um ritmo alarmante. Consulte o mapa do CDC (Centers for Disease Control and Prevention), em www.cdc.gov/obesity/data/adult.html, o qual mostra as tendências de obesidade nos Estados Unidos nos últimos 20 anos. À medida que a obesidade aumenta, o mesmo acontece com os problemas relacionados (por exemplo, doenças do coração, pressão alta, colesterol alto, diabetes tipo 2). Crie um aplicativo que ajude os usuários a escolher ingredientes mais saudáveis ao cozinhar e ajude as pessoas com intolerância a certos alimentos (por exemplo, nozes, glúten) a encontrar substitutos. O aplicativo deve permitir que o usuário digite uma receita e, então, deve sugerir substitutos mais saudáveis para alguns dos ingredientes. Por simplicidade, seu aplicativo deve supor que a receita não tem abreviações para medidas como colher de chá, xícaras e colher de sopa, e deve usar algarismos numéricos para quantidades (por exemplo, 1 ovo, 2 xícaras) em vez de escrevê-las por extenso (um ovo, duas xícaras). Algumas substituições comuns aparecem na Fig. 8.59. Seu aplicativo deve exibir um alerta, como “Consulte sempre seu médico antes de fazer mudanças significativas em sua dieta”.

Ingrediente	Substituição
1 xícara de coalhada	1 xícara de iogurte
1 xícara de leite	1/2 xícara de leite condensado e 1/2 xícara de água
1 colher de chá de suco de limão	1/2 colher de chá de vinagre
1 xícara de açúcar	1/2 xícara de mel, 1 xícara de melado ou 1/4 xícara de xarope de agave
1 xícara de manteiga	1 xícara de margarina ou iogurte
1 xícara de farinha	1 xícara de centeio ou farinha de arroz
1 xícara de maionese	1 xícara de ricota ou 1/8 xícara de maionese e 7/8 xícara de iogurte
1 ovo	2 colheres de sopa de maisena, farinha de araruta ou fécula de batata ou 2 ovos brancos ou 1/2 banana grande (amassada)
1 xícara de leite	1 xícara de leite de soja
1/4 xícara de óleo	1/4 xícara de molho de maçã
pão branco	pão integral

Figura 8.59 Substituições de ingrediente comuns.

O aplicativo deve levar em consideração que nem sempre as substituições são de um para um. Por exemplo, se uma receita de bolo pede três ovos inteiros, em vez disso, pode-se razoavelmente usar seis claras de ovos. Conversão de dados para medidas e substitutos pode ser obtida em sites como:

<http://chinesefood.about.com/od/recipeconversionfaqs/f/usmetricrecipes.htm>
<http://www.pioneerthinking.com/eggsub.html>
<http://www.gourmetsleuth.com/conversions.htm>

Seu aplicativo deve considerar as preocupações do usuário com sua saúde, como colesterol alto, pressão alta, perda de peso, intolerância ao glúten, etc. Para colesterol alto, o aplicativo deve sugerir substitutos para ovos e laticínios; se o usuário quer perder peso, devem ser sugeridos substitutos de baixa caloria para ingredientes como açúcar.

- 8.10 (*Aplicativo Crossword Puzzle Generator*)** Muitas pessoas já fizeram palavras-cruzadas, mas poucas tentaram gerar uma. Crie um aplicativo gerador de palavras-cruzadas pessoais que permita ao usuário digitar as palavras e as dicas correspondentes. Quando ele terminar essa tarefa, gere uma palavra-cruzada usando as palavras fornecidas. Exiba as dicas correspondentes quando o usuário tocar no primeiro quadrado de uma palavra. Se o quadrado representar o início de uma palavra tanto na horizontal como na vertical, mostre as duas dicas.

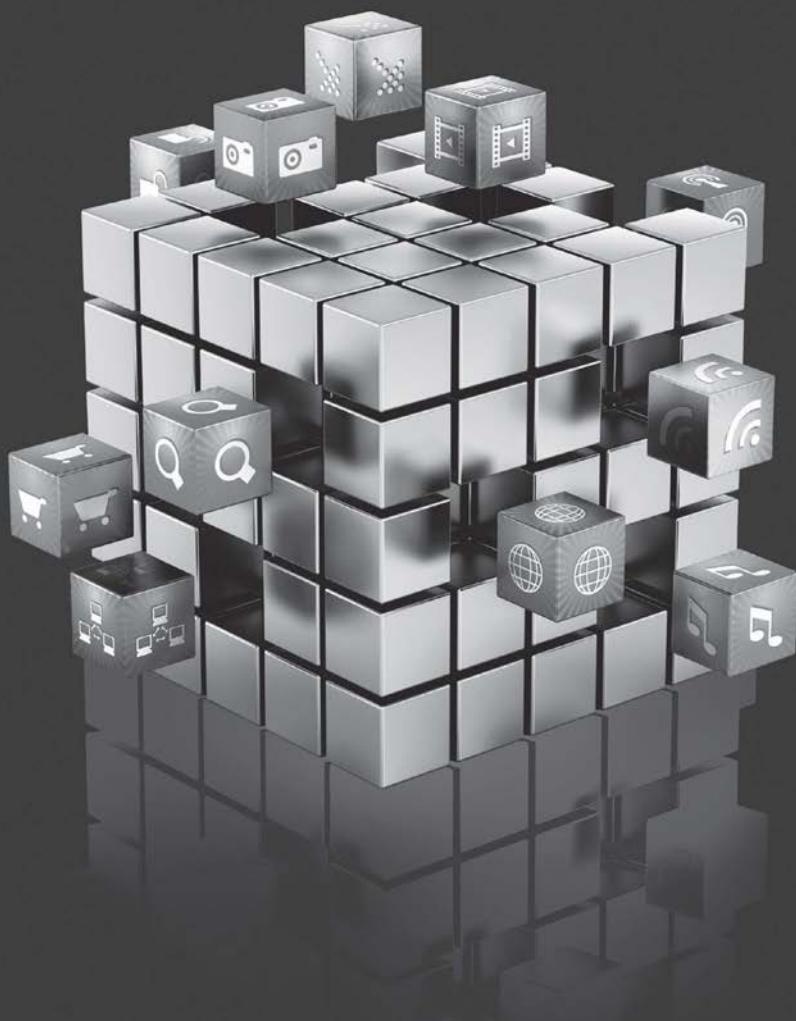
9

Google Play e questões de comercialização de aplicativos

Objetivos

Neste capítulo, você vai:

- Preparar aplicativos para publicação.
- Determinar o preço de seus aplicativos e conhecer as vantagens dos aplicativos gratuitos versus pagos.
- Monetizar seus aplicativos com anúncios incorporados.
- Aprender a vender bens virtuais utilizando cobrança incorporada ao aplicativo.
- Registrar-se no Google Play.
- Aprender a abrir uma conta no Google Wallet.
- Carregar seus aplicativos no Google Play.
- Ativar o **Play Store** dentro de um aplicativo.
- Conhecer outras lojas de aplicativos Android.
- Conhecer outras plataformas populares de aplicativos móveis para as quais você pode portar seus aplicativos a fim de ampliar seu mercado.
- Aprender a comercializar seus aplicativos.



- 9.1 Introdução
- 9.2 Preparação dos aplicativos para publicação
 - 9.2.1 Teste do aplicativo
 - 9.2.2 Acordo de Licença de Usuário Final
 - 9.2.3 Ícones e rótulos
 - 9.2.4 Controlando a versão de seu aplicativo
 - 9.2.5 Licenciamento para controle de acesso a aplicativos pagos
 - 9.2.6 Ofuscando seu código
 - 9.2.7 Obtenção de uma chave privada para assinar digitalmente seu aplicativo
 - 9.2.8 Capturas de tela
 - 9.2.9 Vídeo promocional do aplicativo
- 9.3 Precificação de seu aplicativo: gratuito ou pago
 - 9.3.1 Aplicativos pagos
 - 9.3.2 Aplicativos gratuitos
- 9.4 Monetização de aplicativos com anúncio incorporado
- 9.5 Monetização de aplicativos: utilização de cobrança incorporada para vender bens virtuais
- 9.6 Registro no Google Play
- 9.7 Abertura de uma conta no Google Wallet
- 9.8 Carregamento de seus aplicativos no Google Play
- 9.9 Ativação do **Play Store** dentro de seu aplicativo
- 9.10 Gerenciamento de seus aplicativos no Google Play
- 9.11 Outras lojas de aplicativos Android
- 9.12 Outras plataformas populares de aplicativos móveis
- 9.13 Comercialização de aplicativos
- 9.14 Para finalizar

Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios

9.1 Introdução

Nos Capítulos 2 a 8, desenvolvemos diversos aplicativos Android completos. Uma vez desenvolvidos e testados, tanto no emulador como em dispositivos Android, o próximo passo é enviá-los ao Google Play – e/ou a outras lojas de aplicativos – para distribuição ao público do mundo todo. Neste capítulo, você vai aprender a se registrar no Google Play e a abrir uma conta no Google Wallet para que possa vender seus aplicativos. Também vai aprender a preparar seus aplicativos para publicação e a carregá-los no Google Play. Em alguns casos, vamos encaminhá-lo à documentação do Android, em vez de mostrar os passos no livro, pois é provável que as etapas mudem. Vamos falar sobre outras lojas de aplicativos Android em que você pode distribuir seus aplicativos. Vamos discutir se você deve oferecer seus aplicativos gratuitamente ou vendê-los, e mencionar importantes recursos para monetizar aplicativos, tais como anúncios incorporados e venda de bens virtuais. Vamos fornecer recursos para comercializar seus aplicativos e mencionaremos outras plataformas para as quais você pode portar seus aplicativos Android a fim de ampliar seu mercado.

9.2 Preparação dos aplicativos para publicação

A seção *Preparing for Release* no *Dev Guide* (<http://developer.android.com/tools/publishing/preparing.html>) lista itens a considerar antes de publicar seu aplicativo no Google Play, incluindo:

- *Testar* seu aplicativo em dispositivos Android
- Incluir um *Acordo de Licença de Usuário Final* em seu aplicativo (opcional)
- Adicionar um *ícone* e um rótulo no manifesto do aplicativo
- Fazer o *controle de versão* de seu aplicativo (por exemplo, 1.0, 1.1, 2.0, 2.3, 3.0)

- Obter uma *chave criptográfica* para *assinar digitalmente* seu aplicativo
- *Compilar* seu aplicativo

Antes de publicar seu aplicativo, leia também a *Launch Checklist* (<http://developer.android.com/distribute/googleplay/publish/preparing.html>) e a *Tablet App Quality Checklist* (<http://developer.android.com/distribute/googleplay/quality/tablet.html>).

9.2.1 Teste do aplicativo

Antes de enviar seu aplicativo para o Google Play, teste-o completamente para se certificar de que ele funciona corretamente em uma variedade de dispositivos. Embora o aplicativo possa funcionar perfeitamente usando o emulador em seu computador, podem surgir problemas ao executá-lo em dispositivos Android específicos. Agora o Google Play Developer Console oferece suporte para testes alfa e beta de aplicativos com grupos de pessoas por meio do Google+. Para obter mais informações, visite:

<https://play.google.com/apps/publish/>

9.2.2 Acordo de Licença de Usuário Final

Você tem a opção de incluir um **Acordo de Licença de Usuário Final** (EULA – End User License Agreement) em seu aplicativo. O EULA é um acordo por meio do qual você licencia seu software para o usuário. Normalmente, ele estipula termos de uso, limitações em relação à redistribuição e engenharia reversa, confiabilidade do produto, obediência às leis aplicáveis e muito mais. Talvez você queira consultar um advogado quando esboçar um EULA para seu aplicativo. Para ver um exemplo de EULA, consulte

<http://www.rocketlawyer.com/document/end-user-license-agreement.r1>

9.2.3 Ícones e rótulos

Projete um ícone para seu aplicativo e forneça um rótulo textual (um nome) que aparecerá no Google Play e no dispositivo do usuário. O ícone pode ser o logotipo de sua empresa, uma imagem do aplicativo ou uma imagem personalizada. O Android Asset Studio fornece uma ferramenta para criar ícones de aplicativo:

<http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html>

Crie uma versão de seu ícone para cada uma das seguintes densidades de tela:

- xx-high (XXHDPI): 144 x 144 pixels
- x-high (XHDPI): 96 x 96 pixels
- high (HDPI): 72 x 72 pixels
- medium (MDPI): 48 x 48 pixels

Você também vai precisar de um ícone de alta resolução para usar no Google Play. Esse ícone deve ter:

- 512 x 512 pixels
- PNG de 32 bits
- no máximo 1 MB

Como o ícone é o item mais importante da marca, é fundamental ter um de alta qualidade. Pense na possibilidade de contratar um designer gráfico experiente para ajudá-lo a criar um ícone atraente e profissional. A Figura 9.1 lista diversas empresas de design que oferecem serviços de projeto de ícones profissionais, gratuitos e pagos. Quando tiver criado o ícone e o rótulo, você vai precisar especificá-los no arquivo `AndroidManifest.xml` do aplicativo, configurando os atributos `android:icon` e `android:label` do elemento `application`.

Empresa	URL	Serviços
glyphlab	http://www.glyphlab.com/icon_design/	Projeto de ícones personalizados – alguns ícones podem ser baixados <i>gratuitamente</i> .
Androidicons	http://www.androidicons.com	Projeta ícones personalizados, vende um conjunto de 200 ícones por um valor fixo e alguns ícones podem ser baixados <i>gratuitamente</i> .
Iconiza	http://www.iconiza.com	Projeta ícones personalizados por um valor fixo e vende ícones comuns.
Aha-Soft	http://www.aha-soft.com/icon-design.htm	Projeta ícones personalizados por um valor fixo.
Rosetta®	http://icondesign.rosetta.com/	Projeta ícones personalizados pagos.
Elance®	http://www.elance.com	Busca de designers de ícone freelance.

Figura 9.1 Empresas de projeto de ícones de aplicativo personalizados.

9.2.4 Controlando a versão de seu aplicativo

É importante incluir um *nome* (mostrado para os usuários) e um *código* (um número inteiro, usado internamente pelo Google Play) de versão para seu aplicativo e avaliar sua estratégia de numeração de atualizações. Por exemplo, o primeiro nome de versão de seu aplicativo poderia ser 1.0, pequenas atualizações poderiam ser 1.1 e 1.2, e a próxima atualização importante poderia ser 2.0. O código da versão é um valor inteiro que normalmente começa em 1 e é incrementado por 1 a cada nova versão postada de seu aplicativo. Para ver mais diretrizes, consulte *Versioning Your Applications*, no endereço

<http://developer.android.com/tools/publishing/versioning.html>

9.2.5 Licenciamento para controle de acesso a aplicativos pagos

O *serviço de licenciamento* do Google Play permite criar políticas de licenciamento para controlar o acesso aos seus aplicativos pagos. Por exemplo, você poderia usar uma política de licenciamento para limitar o número de instalações de dispositivo simultâneas permitidas. Para saber mais sobre o serviço de licenciamento, visite:

<http://developer.android.com/google/play/licensing/index.html>

9.2.6 Ofuscando seu código

Você deve “ofuscar” o código dos aplicativos que carregar no Google Play para desencorajar a engenharia reversa e dar maior proteção aos seus aplicativos. A ferramenta gratuita **ProGuard** – executada quando você compila seu aplicativo no *modo release*

– reduz o tamanho de seu arquivo .apk (o arquivo de pacote de aplicativo do Android que contém seu aplicativo para instalação), otimiza e ofusca o código, “removendo código não utilizado e substituindo o nome de classes, campos e métodos por nomes semanticamente obscuros.”¹ Para saber como configurar e usar a ferramenta ProGuard, acesse

<http://developer.android.com/tools/help/proguard.html>

Para obter mais informações sobre como proteger seus aplicativos contra pirataria usando obscurecimento de código, visite:

<http://www.techrepublic.com/blog/app-builder/protect-your-android-apps-with-obfuscation/1724>

9.2.7 Obtenção de uma chave privada para assinar digitalmente seu aplicativo

Antes de carregar seu aplicativo em um dispositivo, no Google Play ou em outras lojas de aplicativos, você deve *assinar digitalmente* o arquivo .apk usando um **certificado digital** que o identifique como autor do aplicativo. Um certificado digital inclui seu nome ou o de sua empresa, informações de contato, etc. Ele pode ser assinado automaticamente usando uma **chave privada** (isto é, uma senha segura usada para *criptografar* o certificado); você não precisa adquirir um certificado de uma autoridade de certificação (embora seja uma opção). O Eclipse assina digitalmente seu aplicativo de forma automática quando você o executa em um emulador ou em um dispositivo para propósitos de *depuração*. Esse certificado digital *não* é válido para uso no Google Play e expira 365 dias depois de ser criado. Para ver instruções detalhadas sobre a assinatura digital de seus aplicativos, consulte *Signing Your Applications* em:

<http://developer.android.com/tools/publishing/app-signing.html>

9.2.8 Capturas de tela

Tire *no mínimo* duas capturas de tela de seu aplicativo (você pode carregar no máximo oito capturas de cada uma para um smartphone, para um tablet de 7” e para um tablet de 10”), as quais vão ser incluídas com a descrição de seu aplicativo no Google Play (Fig. 9.2). Elas oferecem uma visualização prévia de seu aplicativo, pois os usuários não podem testá-lo antes de baixá-lo (embora possam devolver um aplicativo, com reembolso, dentro de 15 minutos após adquiri-lo e baixá-lo). Escolha capturas de tela atraentes que mostrem a funcionalidade do aplicativo.

Especificação	Descrição
Tamanho	Dimensão mínima de 320 pixels e máxima de 3840 pixels (a dimensão máxima não pode ser maior que duas vezes o comprimento da mínima).
Formato	Formato PNG ou JPEG de 24 bits, sem efeitos alfa (transparência).
Imagem	Sangrado total até a margem, sem bordas.

Figura 9.2 Especificações para captura de tela.

¹ <http://developer.android.com/tools/help/proguard.html#enabling>.

O DDMS (Dalvik Debug Monitor Service), instalado com o Plugin ADT para Eclipse, o ajuda a depurar aplicativos em execução em dispositivos reais e também permite fazer capturas de tela em seu dispositivo. Para isso, execute os passos a seguir:

1. Execute o aplicativo em seu dispositivo, conforme descrito no final da Seção 1.9.
2. No Eclipse, selecione **Window > Open Perspective > DDMS**, o que permite usar as ferramentas DDMS.
3. Na janela **Devices** (Fig. 9.3), selecione o dispositivo do qual você deseja obter uma captura de tela.

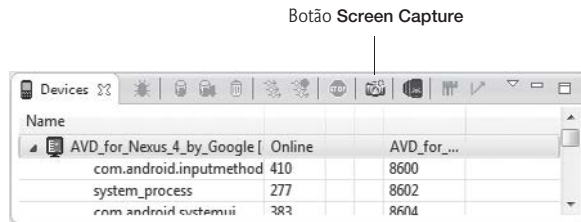


Figura 9.3 Janela **Devices** na perspectiva DDMS.

4. Clique no botão **Screen Capture** para exibir a janela **Device Screen Capture** (Fig. 9.4).

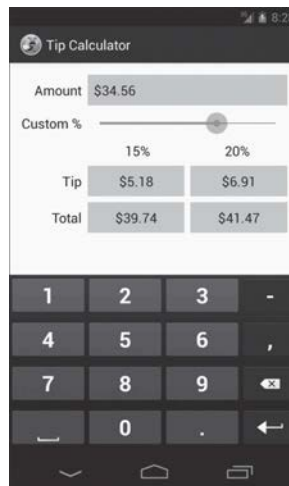


Figura 9.4 Janela **Device Screen Capture** mostrando uma captura do aplicativo **Tip Calculator** do Capítulo 3.

5. Após se certificar de que a tela está mostrando o que você deseja capturar, clique no botão **Save** para salvar a imagem.
6. Se quiser mudar o que está na tela de seu dispositivo antes de salvar a imagem, faça a alteração no dispositivo e, em seguida, pressione o botão **Refresh** na janela **Device Screen Capture** para recapturar a tela do dispositivo.

9.2.9 Vídeo promocional do aplicativo

Quando carregar seu aplicativo no Google Play, você vai ter a opção de incluir uma URL para um breve vídeo promocional no YouTube. A Figura 9.5 lista vários exemplos. Alguns vídeos mostram uma pessoa segurando um dispositivo e interagindo com o aplicativo. Outros utilizam capturas de tela. A Figura 9.6 lista diversas ferramentas e serviços de criação de vídeo (alguns gratuitos, alguns pagos).

Aplicativo	URL
Temple Run®: Oz	http://www.youtube.com/watch?v=QM9sT1ydtj0
GT Racing: Motor Academy	http://www.youtube.com/watch?v=2Z90PICdgoA
Beach Buggy Blitz™	http://www.youtube.com/watch?v=YqDczawTsYw
Real Estate and Homes by Trulia®	http://www.youtube.com/watch?v=rLn697AszGs
Zappos.com®	http://www.youtube.com/watch?v=U-oNyK9k1_Q
Megopolis International	http://www.youtube.com/watch?v=JrqeEJ1xzCY

Figura 9.5 Exemplos de vídeos promocionais para aplicativos no Google Play.

Ferramentas e serviços	URL
Animoto	http://animoto.com
Apptamin	http://www.apptamin.com
Movie Maker for Microsoft Windows	http://windows.microsoft.com/en-us/windows-live/movie-maker
CamStudio™	http://camstudio.org
Jing	http://www.techsmith.com/jing.html
Camtasia Studio®	http://www.techsmith.com/camtasia.html
TurboDemo™	http://www.turbodemo.com/eng/index.php

Figura 9.6 Ferramentas e serviços para criação de vídeos promocionais.

Para carregar seu vídeo, crie uma conta ou entre em sua conta no YouTube. Clique em **Upload** no canto superior direito da página. Clique em **Select files to upload** para escolher um vídeo de seu computador ou simplesmente arraste e solte o arquivo do vídeo na página Web.

9.3 Precificação de seu aplicativo: gratuito ou pago

Você define os preços de seus aplicativos distribuídos por meio do Google Play. Muitos desenvolvedores oferecem seus aplicativos gratuitamente, como uma ferramenta de marketing, publicidade e impressão de marca, lucrando com a venda de produtos e serviços, de versões com mais recursos dos mesmos aplicativos e de conteúdo adicional no aplicativo com *venda incorporada* ou *anúncios incorporados ao aplicativo*. A Figura 9.7 lista maneiras de monetizar seus aplicativos.

Maneiras de monetizar um aplicativo

- Venda do aplicativo no Google Play.
- Venda do aplicativo em outras lojas de aplicativos Android.
- Venda de atualizações pagas.
- Venda de bens virtuais (Seção 9.5).
- Venda de um aplicativo para uma empresa que coloca sua própria marca nele.
- Uso de serviços de propaganda móvel para anúncios incorporados ao aplicativo (Seção 9.4).
- Venda direta ao consumidor de espaço de propaganda incorporado ao aplicativo.
- Uso do aplicativo para impulsionar as vendas de uma versão do aplicativo com mais recursos.

Figura 9.7 Maneiras de monetizar um aplicativo.

9.3.1 Aplicativos pagos

O preço médio dos aplicativos varia muito de acordo com a categoria. Por exemplo, segundo o site de descoberta de aplicativos AppBrain (<http://www.appbrain.com>), o preço médio de aplicativos tipo quebra-cabeça é de US\$1,54, e o de aplicativos comerciais é de US\$6,47.² Embora esses preços possam parecer baixos, lembre-se de que aplicativos de sucesso podem vender dezenas, centenas de milhares, ou mesmo milhões de cópias.

Ao estabelecer um preço para seu aplicativo, comece pesquisando seus concorrentes. Quanto eles cobram? Os aplicativos deles têm funcionalidade semelhante? O seu tem mais recursos? Oferecer seu aplicativo a um preço mais baixo do que o dos concorrentes vai atrair usuários? Seu objetivo é recuperar os custos do desenvolvimento e gerar lucros?

Se mudar de estratégia, você finalmente pode oferecer seu aplicativo como gratuito de forma permanente. Contudo, atualmente não é possível cobrar por um aplicativo que antes era gratuito.

As transações financeiras de aplicativos pagos no Google Play são feitas pelo Google Wallet (<http://google.com/wallet>), embora os clientes de algumas operadoras de telefonia móvel (como AT&T, Sprint e T-Mobile) possam optar por usar a fatura da operadora para cobrar por aplicativos pagos. Os lucros são pagos mensalmente aos usuários do Google Wallet.³ Você é responsável pelos impostos sobre os lucros obtidos por meio do Google Play.

9.3.2 Aplicativos gratuitos

Aproximadamente 80% dos aplicativos no Google Play são gratuitos, sendo responsáveis pela ampla maioria dos downloads.⁴ Visto que é mais provável os usuários baixarem um aplicativo se for gratuito, considere a possibilidade de oferecer uma versão “lite” gratuita de seu aplicativo para estimular os usuários a testá-lo. Por exemplo, se seu aplicativo é um jogo, você poderia oferecer uma versão lite gratuita apenas com os primeiros níveis. Quando o usuário terminasse de jogar os níveis gratuitos, o aplicativo ofereceria uma opção para, pelo Google Play, comprar seu aplicativo mais robusto, com numerosos níveis de jogo. Ou então, exibiria uma mensagem dizendo que o usuário pode comprar mais níveis dentro do aplicativo, em uma atualização transparente (consulte a Seção

² <http://www.appbrain.com/stats/android-market-app-categories>.

³ http://support.google.com/googleplay/android-developer/answer/137997?hl=en&ref_topic=15867.

⁴ <http://www.gartner.com/newsroom/id/2592315>.

9.5). De acordo com um estudo recente feito pela AdMob, *migrar da versão “lite” é o principal motivo de os usuários adquirirem um aplicativo pago*.⁵

Muitas empresas disponibilizam aplicativos gratuitos para difundir o conhecimento da marca e estimular as vendas de outros produtos e serviços (Fig. 9.8).

Aplicativo gratuito	Funcionalidade
Amazon® Mobile	Localizar e adquirir itens no site da Amazon.
Bank of America	Localizar ATMs e agências em sua região, consultar saldos e pagar contas.
Best Buy®	Localizar e adquirir itens.
CNN	Receber as notícias mais recentes do mundo, notícias de última hora e assistir vídeo ao vivo.
Epicurious Recipe	Ver milhares de receitas de várias revistas da Condé Nast, incluindo <i>Gourmet</i> e <i>Bon Appetit</i> .
ESPN® ScoreCenter	Configurar tabelas personalizadas para acompanhar suas equipes esportivas amadoras e profissionais favoritas.
NFL Mobile	Receber as notícias e atualizações mais recentes da NFL, programação ao vivo, NFL Replay e muito mais.
UPS® Mobile	Monitorar cargas, localizar lugares de entrega, obter estimativas de custos de despacho e muito mais.
NYTimes	Ler gratuitamente os artigos do <i>New York Times</i> .
Pocket Agent™	O aplicativo do State Farm Insurance permite que você entre em contato com um agente, faça reclamações, encontre centros de reparos locais, consulte suas contas no State Farm e no fundo mútuo e muito mais.
Progressive® Insurance	Fazer uma reclamação e enviar fotos da cena de um acidente automobilístico, encontrar um agente local, obter informações sobre segurança automobilística quando estiver comprando um carro novo e muito mais.
USA Today®	Ler artigos do <i>USA Today</i> e receber os resultados esportivos mais recentes.
Wells Fargo® Mobile	Localizar ATMs e agências em sua região, consultar saldos, fazer transferências e pagar contas.
Women's Health Workouts Lite	Ver numerosos exercícios físicos de uma das revistas femininas mais importantes.

Figura 9.8 Empresas que disponibilizam aplicativos Android gratuitos para difundir o conhecimento da marca.

9.4 Monetização de aplicativos com anúncio incorporado

Muitos desenvolvedores oferecem aplicativos gratuitos monetizados com **anúncios incorporados** – frequentemente publicidade em banners semelhante às encontradas em sites. Redes de anúncio móvel, como AdMob (<http://www.admob.com/>) e Google AdSense for Mobile (http://www.google.com/mobileads/publisher_home.html), agregam anunciantes para você e colocam anúncios relevantes em seu aplicativo (consulte a Seção 9.13). Você recebe os lucros do anúncio com base no número da métrica *click-through*

⁵ <http://metrics.admob.com/wp-content/uploads/2009/08/AdMob-Mobile-Metrics-July-09.pdf>.

(visualizações). Os 100 aplicativos gratuitos mais vistos podem render desde algumas centenas até milhares de dólares por dia. O anúncio incorporado não gera lucros significativos para a maioria dos aplicativos; portanto, se seu objetivo é recuperar os custos do desenvolvimento e gerar lucros, você deve pensar em cobrar uma taxa por seu aplicativo.

9.5 Monetização de aplicativos: utilização de cobrança incorporada para vender bens virtuais

O serviço de **cobrança incorporada** do Google Play (<http://developer.android.com/google/play/billing/index.html>) permite que você venda **bens virtuais** (por exemplo, conteúdo digital) por meio de aplicativos em dispositivos com Android 2.3 ou superior (Fig. 9.9). De acordo com o Google, os aplicativos que utilizam cobrança incorporada obtêm mais lucro do que os aplicativos pagos sozinhos. Dos 24 jogos que mais geram lucros no Google Play, 23 utilizam cobrança incorporada.⁶ O serviço de cobrança incorporada está disponível apenas para aplicativos adquiridos por meio do Google Play – *não* pode ser usado em aplicativos vendidos em lojas de outros fornecedores. Para usar cobrança incorporada ao aplicativo, você vai precisar de uma conta de publicador no Google Play (consulte a Seção 9.6) e de uma conta no Google Wallet (consulte a Seção 9.7). O Google paga a você 70% dos lucros de todas as compras incorporadas feitas por meio dos seus aplicativos.

Bens virtuais		
Assinaturas eletrônicas de revistas	Guias adaptados ao idioma local	Avatares
Vestuário virtual	Níveis de jogo adicionais	Cenário de jogo
Recursos complementares	Tons de chamada	Ícones
Cartões eletrônicos	Presentes eletrônicos	Moeda virtual
Papéis de parede	Imagens	Animais de estimação virtuais
Áudios	Vídeos	Livros eletrônicos e muito mais

Figura 9.9 Bens virtuais.

A venda de bens virtuais pode gerar lucros mais altos *por usuário* do que os anúncios incorporados ao aplicativo.⁷ Alguns aplicativos particularmente bem-sucedidos na venda de bens virtuais incluem Angry Birds, DragonVale, Zynga Poker, Bejeweled Blitz, NYTimes e Candy Crush Saga. Os bens virtuais são especialmente populares em jogos móveis.

Para implementar cobrança incorporada ao aplicativo, siga os passos que se encontram em

http://developer.android.com/google/play/billing/billing_integrate.html

Para obter mais informações sobre cobrança incorporada ao aplicativo, incluindo assinaturas, amostras de aplicativo, melhores práticas de segurança, testes e muito mais, visite http://developer.android.com/google/play/billing/billing_overview.html. Também é possível assistir à aula de treinamento gratuita *Selling In-app Products* em

<http://developer.android.com/training/in-app-billing/index.html>

⁶ <http://android-developers.blogspot.com/2012/05/in-app-subscriptions-in-google-play.html>.

⁷ http://www.businessinsider.com/its-morning-in-venture-capital-2012-5?utm_source=readme&utm_medium=rightrail&utm_term=&utm_content=6&utm_campaign=recirc.

Aquisição incorporada para aplicativos vendidos por meio de outras lojas

Se você optar por vender seus aplicativos por meio de outras lojas (consulte a Seção 9.11), vários provedores de pagamento móvel permitem que seus aplicativos tenham *aquisição incorporada* usando APIs de provedores de pagamento móvel (Fig. 9.10) – você não pode usar a cobrança incorporada ao aplicativo do Google Play. Comece incorporando *funcionalidades bloqueadas* adicionais (por exemplo, níveis de jogo, avatares) em seu aplicativo. Quando o usuário optar por fazer uma compra, a ferramenta de aquisição incorporada ao aplicativo cuida da transação financeira e retorna uma mensagem para o aplicativo confirmando o pagamento. Então, o aplicativo desbloqueia a funcionalidade adicional. As operadoras de telefonia móvel recolhem entre 25% e 45% do preço.*

Provedor	URL	Descrição
PayPal Mobile Payments Library	http://developer.paypal.com/webapps/developer/docs/classic/mobile/gs_MPL/	Os usuários clicam no botão Pay with PayPal , fazem login em sua conta no PayPal e, então, clicam em Pay .
Amazon In-App Purchasing	http://developer.amazon.com/sdk/in-app-purchasing.html	Aquisição incorporada para aplicativos vendidos por meio da Amazon App Store para Android.
Zong	http://www.zong.com/android	Fornecer um botão Buy para pagamento com um clique. Os pagamentos aparecem na conta telefônica do usuário.
Samsung In-App Purchase	http://developer.samsung.com/android/tools-sdks/In-App-Purchase-Library	Aquisição incorporada para aplicativos projetados especificamente para dispositivos Samsung.
Boku	http://www.boku.com	Os usuários clicam em Pay by Mobile , digitam o número de seus celulares e, então, completam a transação respondendo a uma mensagem de texto enviada para seus telefones.

Figura 9.10 Provedores de pagamento móvel para aquisição incorporada ao aplicativo.

9.6 Registro no Google Play

Para publicar seus aplicativos no Google Play, você deve registrar uma conta no endereço

<http://play.google.com/apps/publish>

Há uma taxa de registro de US\$25, paga somente uma vez. Ao contrário de outras plataformas móveis populares, *o Google Play não tem processo algum de aprovação para o envio de aplicativos para a loja*. Contudo, você precisa aceitar as *Google Play Developers Program Policies*. Se seu aplicativo violar essas políticas, poderá ser removido a qualquer momento. Violações graves ou repetidas podem resultar no encerramento da conta (Fig.9.11).

Violações do Google Play Content Policy for Developers

- Infringir os direitos de propriedade intelectual de outros (por exemplo, marcas registradas, patentes e direitos autorais).
- Atitudes ilegais.
- Invadir privacidade pessoal.
- Interferir nos serviços de terceiros.
- Danificar o dispositivo ou os dados pessoais do usuário.

Figura 9.11 Algumas violações do Google Play Content Policy for Developers (<http://play.google.com/about/developer-content-policy.html#showlanguages>). (continua)

* N. de T. Estes percentuais são para os Estados Unidos.

Violações do Google Play Content Policy for Developers

- Jogos de azar.
- Criar experiência de usuário tipo “spam” (por exemplo, enganar o usuário sobre o propósito do aplicativo).
- Causar impacto negativo nas taxas de serviço do usuário ou na rede da operadora de telefonia sem fio.
- Personificação ou fraude.
- Promover ódio ou violência.
- Fornecer conteúdo pornográfico ou obsceno, ou qualquer coisa inadequada a crianças com menos de 18 anos.
- Anúncios em notificações e widgets em nível de sistema.

Figura 9.11 Algumas violações do Google Play Content Policy for Developers (<http://play.google.com/about/developer-content-policy.html#showlanguages>).

9.7 Abertura de uma conta no Google Wallet

Para vender seus aplicativos no Google Play, você vai precisar de uma **conta de vendedor (merchant account) no Google Wallet**, disponível para desenvolvedores de Google Play em 32 países (Fig. 9.12).⁸ O Google Wallet é usado como serviço de pagamento para transações online. Quando tiver se registrado e conectado no Google Play, em <http://play.google.com/apps/publish/>, clique no link **Financial Reports** e, então, clique em **Set up a merchant account**. Você precisará:

- fornecer informações privadas para o Google entrar em contato com você;
- fornecer informações de contato de suporte ao cliente para que os usuários possam contatá-lo;
- fornecer informações financeiras para que o Google possa fazer uma verificação de crédito;
- concordar com os termos de serviço, os quais descrevem as características do serviço, as transações permitidas, as ações proibidas, as taxas de serviço, os termos de pagamento e muito mais.

Países			
Alemanha	Dinamarca	Irlanda	Portugal
Argentina	Espanha	Israel	Reino Unido
Austrália	Estados Unidos	Itália	República Checa
Áustria	Finlândia	Japão	Rússia
Bélgica	França	México	Singapura
Brasil	Holanda	Noruega	Suécia
Canadá	Hong Kong	Nova Zelândia	Suíça
Coreia do Sul	Índia	Polônia	Taiwan

Figura 9.12 Países nos quais estão disponíveis as contas de vendedor no Google Wallet.

O Google Wallet processa os pagamentos e o ajuda a se proteger contra compras fraudulentas. As taxas de processamento de pagamento padrão são devolvidas mediante suas vendas no Google Play.⁹ O Google paga a você 70% do preço do aplicativo. Uma

⁸ http://support.google.com/googleplay/android-developer/answer/150324?hl=en&ref_topic=15867.

⁹ <http://checkout.google.com/termsOfService?type=SELLER>.

vez aberta a conta no Google Wallet, você pode utilizá-la para mais atividades além de apenas vender seus aplicativos, como fazer compras nas lojas participantes.

9.8 Carregamento de seus aplicativos no Google Play

Uma vez que você tenha preparado os seus arquivos e esteja pronto para carregar o aplicativo para a loja, examine os passos na *Launch Checklist*, no endereço:

<http://developer.android.com/distribute/googleplay/publish/preparing.html>

Depois, faça login no Google Play em <http://play.google.com/apps/publish> (Seção 9.6) e clique no botão **Publish an Android App on Google Play** para iniciar o processo de carregamento. Será solicitado que você faça upload dos seguintes itens:

1. O *arquivo .apk do aplicativo*, que inclui os arquivos de código do aplicativo, itens, recursos e o arquivo de manifesto.
2. Pelo menos *duas capturas de tela* de seu aplicativo, para serem incluídas no Google Play. Você pode incluir capturas de tela para um telefone Android, para um tablet de 7" e para um tablet de 10".
3. *Ícone de aplicativo de alta resolução* (512 x 512 pixels) para ser incluído no Google Play.
4. *Elemento gráfico promocional* (opcional) do Google Play para ser usado pelo Google caso eles decidam promover seu aplicativo (para ver exemplos, veja alguns dos elementos gráficos de aplicativos especiais no Google Play). O elemento gráfico deve ter 180 pixels de largura x 120 pixels de altura, no formato PNG ou JPEG de 24 bits, *sem efeitos de transparência alfa*. Também deve ter sangrado total (isto é, ir até a margem da tela, sem qualquer borda no elemento gráfico).
5. *Vídeo promocional* (opcional) para ser incluído no Google Play. Você pode incluir uma URL para um vídeo promocional de seu aplicativo (por exemplo, um link no YouTube para um vídeo demonstrando o funcionamento do aplicativo).

Além dos itens do aplicativo, será solicitado que você forneça a seguinte listagem adicional de detalhes para o Google Play:

1. **Language (idioma)**. Por padrão, seu aplicativo vai ser listado em inglês. Se quiser listá-lo em mais idiomas, selecione-os na relação fornecida (Fig. 9.13).

Idioma				
Africâner	Alemão	Aramaico	Árabe	Bielorrusso
Catalão	Chinês (simplificado ou tradicional)	Coreano	Croata	Dinamarquês
Eslovaco	Esloveno	Espanhol (Am. Latina, Espanha ou EUA)	Estoniano	Filipino
Finlandês	Francês	Grego	Hebraico	Holandês
Húngaro	Indonésio	Inglês (RU ou EUA)	Italiano	Japonês
Letão	Lituano	Malaio	Norueguês	Persa

Figura 9.13 Idiomas para listar aplicativos no Google Play. (continua)

Idioma				
Polonês	Português (Brasil ou Portugal)	Romanche	Romeno	Russo
Sérvio	Suaíli	Suíço	Tailandês	Tcheco
Turco	Ucraniano	Urdu	Vietnamita	Zulu

Figura 9.13 Idiomas para listar aplicativos no Google Play.

2. **Title (Título).** O título de seu aplicativo, conforme vai aparecer no Google Play (30 caracteres no máximo). *Não* precisa ser único entre todos os aplicativos Android.
3. **Description (Descrição).** Uma descrição do aplicativo e seus recursos (4.000 caracteres no máximo). Recomenda-se usar a última parte da descrição para explicar por que cada permissão é exigida e como é usada.
4. **Recent changes (Alterações recentes).** Um acompanhamento de quaisquer alterações específicas na versão mais recente de seu aplicativo (500 caracteres no máximo).
5. **Promo text (Texto promocional).** O texto promocional para comercializar seu aplicativo (80 caracteres no máximo).
6. **App type (Tipo de aplicativo).** Escolha Applications ou Games.
7. **Category (Categoria).** Selecione a categoria (consulte a Fig. 1.8) mais adequada ao seu jogo ou aplicativo.
8. **Price (Preço).** A configuração padrão é Free. Para vender seu aplicativo, você vai precisar de uma conta de vendedor no Google Wallet.
9. **Content rating (Classificação de conteúdo).** Você pode selecionar High Maturity, Medium Maturity, Low Maturity ou Everyone. Para obter mais informações, consulte *Rating your application content for Google Play* em <http://support.google.com/googleplay/android-developer/answer/188189>.
10. **Locations (Locais).** Por padrão, o aplicativo vai ser listado em todos os países abrangidos pelo Google Play, atuais e futuros. Se não quiser que seu aplicativo esteja disponível em todos esses países, escolha cuidadosamente aqueles específicos em que deseja listá-lo.
11. **Website (Site).** Um link Visit Developer's Website será incluído na listagem de seu aplicativo no Google Play. Forneça um link direto para a página em seu site, em que os usuários interessados em baixar seu aplicativo possam encontrar mais informações, incluindo material de marketing, listagens de recursos, mais capturas de tela, instruções, etc.
12. **E-mail (E-mail).** Seu endereço de e-mail também vai ser incluído no Google Play, para que os clientes possam entrar em contato com você para fazer perguntas, relatar erros, etc.
13. **Phone number (Número de telefone).** Às vezes seu número de telefone é incluído no Google Play. Portanto, recomenda-se deixar esse campo em branco, a não ser que você dê suporte via telefone. Talvez você queira fornecer o número do telefone do atendimento ao cliente em seu site.

9.9 Ativação do Play Store dentro de seu aplicativo

Para estimular suas vendas, você pode ativar o aplicativo **Play Store** (Google Play) dentro de seu aplicativo (normalmente, incluindo um botão) para que o usuário possa

baixar outros aplicativos seus publicados ou comprar um aplicativo relacionado com funcionalidade maior do que a da versão “lite” baixada anteriormente. Você também pode ativar o aplicativo **Play Store** para permitir que os usuários baixem as atualizações mais recentes.

Existem duas maneiras de ativar o aplicativo **Play Store**. Primeiramente, você pode pesquisar no Google Play em busca de aplicativos com um nome de desenvolvedor, nome de pacote ou uma string de caracteres específicos. Por exemplo, se quiser estimular os usuários a baixar outros aplicativos seus publicados, você pode incluir em seu aplicativo um botão que, quando tocado, ativa o aplicativo **Play Store** e inicia uma busca por aplicativos contendo o seu nome ou o de sua empresa. A segunda opção é levar o usuário para a página de detalhes no aplicativo **Play Store** a fim de buscar um aplicativo específico.

Para saber como ativar o **Play Store** dentro de um aplicativo, consulte *Linking Your Products* em <http://developer.android.com/distribute/googleplay/promote/linking.html>.

9.10 Gerenciamento de seus aplicativos no Google Play

O *Google Play Developer Console* permite gerenciar sua conta e seus aplicativos, verificar as pontuações em estrelas dadas pelos usuários para seus aplicativos (0 a 5 estrelas), responder aos comentários dos usuários e monitorar o número total de instalações de cada aplicativo e o número de instalações ativas (instalações menos desinstalações). Você pode ver as tendências de instalação e a distribuição de downloads de aplicativo pelas versões de Android, dispositivos e muito mais. Crash reports lista qualquer informação de falha e congelamento dada pelos usuários. Caso tenha feito atualizações em seu aplicativo, você pode publicar a nova versão facilmente. Você pode retirar o aplicativo do Google Play, mas os usuários que o tiverem baixado anteriormente podem mantê-lo em seus dispositivos. Os usuários que desinstalaram o aplicativo poderão reinstalá-lo mesmo após ele ser removido (ele vai permanecer nos servidores do Google, a não ser que seja removido por violar os termos de serviço).

9.11 Outras lojas de aplicativos Android

Além do Google Play, você pode optar por disponibilizar seus aplicativos em outras lojas de aplicativos Android (Fig. 9.14) ou mesmo em seu próprio site, usando serviços como AndroidLicenser (<http://www.androidlicenser.com>). Para saber mais sobre como lançar seu aplicativo por meio de um site, consulte

http://developer.android.com/tools/publishing/publishing_overview.html

Loja	URL
Amazon Appstore	http://developer.amazon.com/welcome.html
Opera Mobile Store	http://apps.opera.com/en_us/index.php
Moborobo	http://www.moborobo.com
Appitalism®	http://www.appitalism.com/index.html
Samsung Apps	http://apps.samsung.com/mars/main/getMain.as

Figura 9.14 Outras lojas de aplicativos Android. (continua)

Loja	URL
GetJar	http://www.getjar.com
SlideMe	http://www.slideme.org
Handango	http://www.handango.com
Mplayit™	http://www.mplayit.com
AndroidPIT	http://www.androidpit.com

Figura 9.14 Outras lojas de aplicativos Android.

9.12 Outras plataformas populares de aplicativos móveis

De acordo com a ABI Research, 56 bilhões de aplicativos para smartphone e 14 bilhões de aplicativos para tablet iriam ser baixados em 2013.¹⁰ Portando seus aplicativos Android para outras plataformas móveis, especialmente para o iOS (para dispositivos iPhone, iPad e iPod Touch), você poderia atingir um público ainda maior (Fig. 9.15). O Android pode ser desenvolvido em computadores Windows, Linux ou Mac com Java – uma das linguagens de programação mais usadas do mundo. Contudo, os aplicativos para iOS devem ser desenvolvidos em Macs – que são caros – e com a linguagem de programação Objective-C, a qual apenas uma pequena porcentagem dos desenvolvedores conhece. O Google criou a ferramenta de código-fonte aberto J2ObjC a fim de ajudar a traduzir seu código de aplicativo Java em Objective-C para aplicativos iOS. Para saber mais, consulte <http://code.google.com/p/j2objc/>.

Plataforma	URL	Fatia de mercado do download de aplicativos no mundo todo
Android	http://developer.android.com	58% de aplicativos para smartphone 17% de aplicativos para tablet
iOS (Apple)	http://developer.apple.com/ios	33% de aplicativos para smartphone 75% de aplicativos para tablet
Windows Phone 8	http://developer.windowsphone.com	4% de aplicativos para smartphone 2% de aplicativos para tablet
BlackBerry (RIM)	http://developer.blackberry.com	3% de aplicativos para smartphone
Amazon Kindle	http://developer.amazon.com	4% de aplicativos para tablet

Fig. 9.15 Plataformas populares de aplicativos móveis. (<http://www.abiresearch.com/press/android-will-account-for-58-of-smartphone-app-down>).

9.13 Comercialização de aplicativos

Uma vez publicado seu aplicativo, você vai querer comercializá-lo para seu público.¹¹ O *marketing viral* por meio de sites de mídia social, como Facebook, Twitter, Google+ e YouTube, pode ajudá-lo a transmitir sua mensagem. Esses sites têm enorme visibilidade. De acordo com um estudo do Pew Research Center, 72% dos adultos na Internet

¹⁰ <http://www.abiresearch.com/press/android-will-account-for-58-of-smartphone-app-down>.

¹¹ Para aprender mais sobre comercialização de aplicativos Android, consulte o livro *Android Apps Marketing: Secrets to Selling Your Android App*, de Jeffrey Hughes.

utilizam redes sociais – e 67% deles estão no Facebook.¹² A Figura 9.16 lista alguns dos sites de mídia social mais populares. Além disso, e-mail e boletins eletrônicos ainda são ferramentas de marketing eficazes e frequentemente baratas.

Nome	URL	Descrição
Facebook	http://www.facebook.com	Rede social
Twitter	http://www.twitter.com	Microblog, rede social
Google+	http://plus.google.com	Rede social
Groupon	http://www.groupon.com	Comércio eletrônico
Foursquare	http://www.foursquare.com	Check-in
Pinterest	http://www.pinterest.com	Quadro de avisos online
YouTube	http://www.youtube.com	Compartilhamento de vídeos
LinkedIn	http://www.linkedin.com	Rede social para negócios
Flickr	http://www.flickr.com	Compartilhamento de fotos

Figura 9.16 Sites de mídia social populares.

Facebook

O Facebook, o mais importante site de rede social, tem mais de 1 bilhão de usuários ativos¹³ e mais de 150 bilhões de conexões entre amigos.¹⁴ É um excelente recurso para *marketing viral*. Comece criando uma página oficial para seu aplicativo ou sua empresa no Facebook. Use a página para postar informações sobre o aplicativo, novidades, atualizações, análises, dicas, vídeos, capturas de tela, pontuações mais altas de jogos, opinião dos usuários e links para o Google Play, onde eles podem baixar seu aplicativo. Por exemplo, postamos notícias e atualizações sobre as publicações da Deitel em nossa página no Facebook, no endereço <http://www.facebook.com/DeitelFan>.

Em seguida, você precisa difundir a notícia. Estimule seus colegas e amigos a “curtir” sua página no Facebook e peça aos amigos deles para fazerem o mesmo. À medida que as pessoas interagirem com sua página, histórias vão aparecer nos feeds de notícias de seus amigos, divulgando o conhecimento para um público crescente.

Twitter

O Twitter é um site de microblog e rede social com mais de 554 milhões de usuários registrados ativos.¹⁵ Você posta tweets – mensagens de 140 caracteres ou menos. Então, o Twitter distribui seus tweets para todos os seus seguidores (quando este livro estava sendo produzido, um cantor famoso tinha mais de 40 milhões de seguidores). Muitas pessoas usam o Twitter para monitorar notícias e tendências. Envie tweets sobre seu aplicativo – inclua anúncios sobre novos lançamentos, dicas, informações, comentários dos usuários, etc. Estimule também seus colegas e amigos a enviar um tweet sobre seu aplicativo. Use uma *hashtag* (#) para referenciar seu aplicativo. Por exemplo, ao enviar um tweet sobre este livro em nosso feed no Twitter, @deitel, usamos a hashtag #AndroidHTP2. Outros também podem usar essa hashtag para escrever comentários sobre o livro. Isso permite que você procure facilmente tweets de mensagens relacionadas ao livro.

¹² <http://pewinternet.org/Commentary/2012/March/Pew-Internet-Social-Networking-full-detail.aspx>.

¹³ <http://investor.fb.com/releasedetail.cfm?ReleaseID=761090>.

¹⁴ <http://expandedramblings.com/index.php/by-the-numbers-17-amazing-facebook-stats/>.

¹⁵ <http://www.statisticbrain.com/twitter-statistics/>.

Vídeo viral

O vídeo viral – compartilhado em sites de vídeo (por exemplo, YouTube, Bing Videos, Yahoo! Video), em sites de rede social (por exemplo, Facebook, Twitter e Google+), por meio de e-mail, etc. – é outra excelente maneira de divulgar seu aplicativo. Se você criar um vídeo cativante, talvez humorístico ou mesmo escandaloso, ele pode ganhar popularidade rapidamente e ser marcado por usuários de várias redes sociais.

Newsletter por e-mail

Caso você tenha uma newsletter por e-mail, utilize-a para promover seu aplicativo. Inclua links para o Google Play, onde os usuários podem baixar o aplicativo. Inclua também links para suas páginas de rede social, onde os usuários podem permanecer atualizados com as notícias mais recentes sobre seu aplicativo.

Análises de aplicativo

Entre em contato com blogs e sites de análise de aplicativo influentes (Fig. 9.17) e informe-os sobre seu aplicativo. Forneça a eles um código promocional para baixar seu aplicativo gratuitamente (consulte a Seção 9.3). Blogueiros e analistas influentes recebem muitos pedidos; portanto, mantenha o seu conciso e informativo, sem muita propaganda. Muitos analistas de aplicativo postam análises em vídeo no YouTube e em outros sites (Fig. 9.18).

Site de análise de aplicativos Android	URL
Android Tapp™	http://www.androidtapp.com
Appolicious™	http://www.androidapps.com
AppBrain	http://www.appbrain.com
AndroidZoom	http://www.androidzoom.com
Appstorm	http://android.appstorm.net
Best Android Apps Review	http://www.bestandroidappsreview.com
Android App Review Source	http://www.androidappreviewsources.com
Androinica	http://www.androinica.com
AndroidLib	http://www.androidlib.com
Android and Me	http://www.androidandme.com
AndroidGuys	http://www.androidguys.com/category/reviews
Android Police	http://www.androidpolice.com
AndroidPIT	http://www.androidpit.com
Phandroid	http://www.phandroid.com

Figura 9.17 Sites de análise de aplicativos Android.

Site com vídeo de análise de aplicativos Android	URL
Daily App Show	http://dailyappshow.com
Crazy Mike's Apps	http://crazymikesapps.com
Appolicious™	http://www.appvee.com/?device_filter=android
Life of Android™	http://www.lifeofandroid.com/video/
Android Video Review	http://www.androidvideoreview.net/

Figura 9.18 Sites com vídeo de análise de aplicativos Android.

Relações públicas na Internet

O setor de relações públicas utiliza os meios de comunicação para ajudar as empresas a transmitir sua mensagem aos consumidores. Com o fenômeno conhecido como Web 2.0, os profissionais de relações públicas estão incorporando blogs, tweets, podcasts, feeds RSS e mídia social em suas campanhas. A Figura 9.19 lista alguns recursos de relações públicas na Internet, gratuitos e pagos, incluindo sites de distribuição de press release, serviços de redação de press release e muito mais.

Recurso de relações públicas na Internet	URL	Descrição
Serviços gratuitos		
PRWeb®	http://www.prweb.com	Serviços de distribuição de press release online, <i>gratuitos</i> e <i>pagos</i> .
ClickPress™	http://www.clickpress.com	Envie novas matérias para aprovação (<i>grátis</i>). Se forem aprovadas, elas estarão disponíveis no site da ClickPress e nos mecanismos de busca de notícias. Mediante uma <i>taxa</i> , a ClickPress distribui seus press releases em escala global para as principais revistas online de economia.
PRLog	http://www.prlog.org/pub/	Envio e distribuição <i>gratuitos</i> de press release.
i-Newswire	http://www.i-newswire.com	Envio e distribuição <i>gratuitos</i> e <i>pagos</i> de press release.
openPR®	http://www.openpr.com	Publicação <i>gratuita</i> de press release.
Serviços pagos		
PR Leap	http://www.prleap.com	Serviço de distribuição de press release online.
Marketwire	http://www.marketwire.com	O serviço de distribuição de press release permite a você atingir seu público por área geográfica, segmento, etc.
Mobility PR	http://www.mobilitypr.com	Serviços de relações públicas para empresas no setor de equipamentos móveis.
Press Release Writing	http://www.press-release-writing.com	Distribuição de press release e serviços que incluem redação, revisão e edição de press release. Consulte as dicas para redigir press releases eficazes.

Figura 9.19 Recursos de relações públicas na Internet.

Redes de anúncios móveis

Comprar espaços de publicidade (por exemplo, em outros aplicativos, online, em jornais e revistas ou no rádio e na televisão) é outra maneira de comercializar seu aplicativo. As redes de anúncios móveis (Fig. 9.20) são especializadas em divulgar aplicativos Android (e outros) móveis em plataformas móveis. Muitas dessas redes podem atingir o público de acordo com o local, operadora de telefonia sem fio, plataforma (por exemplo, Android, iOS, Windows, BlackBerry) e muito mais. A maioria dos aplicativos não gera muito dinheiro; portanto, cuidado com quanto vai gastar em anúncios.

Redes de anúncios móveis	URL
AdMob (da Google)	http://www.admob.com/
Medialets	http://www.medialets.com
Tapjoy®	http://www.tapjoy.com
Nexage™	http://www.nexage.com
JumpTap®	http://www.jumptap.com
Smaato®	http://www.smaato.com
mMedia™	http://mmedia.com
InMobi™	http://www.inmobi.com
Flurry™	http://www.flurry.com

Figura 9.20 Redes de anúncios móveis.

Você também pode usar essas redes de anúncios móveis para monetizar seus aplicativos gratuitos incluindo anúncios (por exemplo, banners, vídeos) em seus aplicativos. O eCPM (custo efetivo por 1.000 impressões) médio para anúncios em aplicativos Android é de US\$0,88, segundo o relatório *State of Mobile Advertising* do Opera¹⁶ (embora a média possa variar de acordo com a rede, dispositivo, etc.). A maioria dos anúncios no Android é paga com base na *taxa de click-through (CTR)* dos anúncios, em vez de no número de impressões geradas. Segundo um relatório da JumpTap, as CTRs atingem uma média de 0,65% em anúncios incorporados aos aplicativos móveis,¹⁷ embora isso varie de acordo com o aplicativo, com o dispositivo, com o alvo dos anúncios por rede e muito mais. Se seu aplicativo tem muitos usuários e as CTRs dos anúncios em seus aplicativos são altas, você pode lucrar muito com publicidade. Além disso, sua rede de anúncios pode render publicidade de valor mais alto, aumentando assim seus lucros.

9.14 Para finalizar

Neste capítulo, explicamos o processo de registro do Google Play e a abertura de uma conta no Google Wallet para que você possa vender seus aplicativos. Mostramos como preparar aplicativos para enviar ao Google Play, incluindo testá-los no emulador e em dispositivos Android, criar ícones e rótulos, e editar o arquivo `AndroidManifest.xml`. Explicamos os passos necessários para carregar seus aplicativos no Google Play. Mostramos lojas de aplicativos Android alternativas em que você pode vender seus aplicativos. Fornecemos dicas para determinar o preço de seus aplicativos e recursos para monetizá-los com anúncio incorporado e vendas de bens virtuais no aplicativo. Além disso, incluímos recursos para comercializar seus aplicativos quando estiverem disponíveis por meio do Google Play.

Mantenha contato com a Deitel & Associates, Inc.

Esperamos que você tenha gostado de ler este livro tanto quanto nós gostamos de escrevê-lo. Gostaríamos de saber sua opinião. Envie suas perguntas, comentários, sugestões e correções para deitel@deitel.com. Consulte nossa lista de Resource Centers relacionados ao Android em <http://www.deitel.com/ResourceCenters.html>. Para manter-se

¹⁶ <http://www.insidemobileapps.com/2012/12/14/ios-leads-the-pack-in-ecpm-traffic-and-revenue-on-operas-mobile-ad-platform-ipad-average-ecpm-of-4-42/>.

¹⁷ <http://paidcontent.org/2012/01/05/419-jumtap-android-the-most-popular-but-ios-still-more-interactive-for-ads/>.

atualizado em relação às notícias mais recentes sobre as publicações e os treinamentos corporativos da Deitel, assine nossa newsletter semanal – enviada gratuitamente por e-mail –, *Deitel® Buzz Online*, em

<http://www.deitel.com/newsletter/subscribe.html>

e siga-nos no

- Facebook – <http://www.facebook.com/DeitelFan>
- Twitter – @deitel
- Google+ – <http://google.com/+DeitelFan>
- YouTube – <http://youtube.com/DeitelTV>
- LinkedIn – <http://linkedin.com/company/deitel-&-associates>

Para saber mais sobre o treinamento em programação *in loco* da Deitel & Associates em todo o mundo, para sua empresa ou organização, visite:

<http://www.deitel.com/training>

ou mande um e-mail para deitel@deitel.com.

Exercícios de revisão

9.1 Preencha os espaços em branco em cada um dos seguintes enunciados:

- a) Para vender seus aplicativos no Google Play, você precisará de uma conta no _____.
- b) Antes de carregar seu aplicativo em um dispositivo, no Google Play ou em outras lojas de aplicativos, você deve assinar digitalmente o arquivo .apk (arquivo de pacote de aplicativo do Android) usando um _____ que o identifique como autor do aplicativo.
- c) O _____ do Google Play permite gerenciar sua conta e seus aplicativos, verificar as pontuações em estrelas dadas pelos usuários para seus aplicativos (0 a 5 estrelas), monitorar o número total de instalações de cada aplicativo e o número de instalações ativas (instalações menos desinstalações).

9.2 Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.

- a) Quando um aplicativo funcionar perfeitamente usando o emulador em seu computador, ele funcionará em seu dispositivo Android.
- b) Você poderia usar uma política de licenciamento para limitar a frequência com que o aplicativo é transferido para o servidor, quantas instalações simultâneas de dispositivo são permitidas e o que acontece quando um aplicativo não licenciado é identificado.
- c) O título de seu aplicativo, conforme vai aparecer no Google Play, deve ser único dentre todos os aplicativos Android.
- d) De acordo com um estudo da empresa de análise de lojas de aplicativo Distimo (www.distimo.com/), o preço médio de aplicativos de jogo Android pagos gira em torno de US\$36.20.
- e) De acordo com o Google, os aplicativos que utilizam cobrança incorporada obtêm mais lucro do que os aplicativos pagos sozinhos.
- f) Se você optar por vender seus aplicativos por meio de outras lojas, vários provedores de pagamento móvel permitem que seus aplicativos tenham aquisição incorporada usando APIs de provedores de pagamento móvel.

Respostas dos exercícios de revisão

- 9.1** a) Google Wallet. b) certificado digital. c) Developer Console.
- 9.2** a) Falsa. Embora o aplicativo possa funcionar perfeitamente usando o emulador em seu computador, podem surgir problemas ao executá-lo em um dispositivo Android específico. b) Verdadeira. c) Falsa. O título de seu aplicativo, conforme vai aparecer no Google Play, *não* precisa ser único dentre todos os aplicativos Android. d) Falsa. De acordo com o estudo, o preço médio de aplicativos de jogo gira em torno de US\$3.27 (a mediana gira em torno de US\$2.72). e) Verdadeira. f) Verdadeira.

Exercícios

- 9.3** Preencha os espaços em branco em cada um dos seguintes enunciados:
- _____ exibem informações oportunas na tela inicial do usuário, como o clima, preços de ações e notícias.
 - O _____ é um acordo por meio do qual você licencia seu software para o usuário. Normalmente, ele estipula termos de uso, limitações em relação à redistribuição e engenharia reversa, confiabilidade do produto, obediência às leis aplicáveis e muito mais.
 - O _____, que é instalado com o Plugin ADT para Eclipse, o ajuda a depurar aplicativos em execução em dispositivos reais.
 - De acordo com um estudo recente da AdMob, _____ é o principal motivo pelo qual os usuários compram um aplicativo.
 - A _____ do Google Play possibilita limitar o número de instalações simultâneas de dispositivo permitidas.
- 9.4** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- Você deve “ofuscar” os aplicativos que carregar no Google Play para estimular a engenharia reversa de seu código.
 - Existem mais aplicativos pagos do que gratuitos no Google Play, e eles compreendem a ampla maioria dos downloads.
 - O nome da versão de seu aplicativo é mostrado para os usuários, e o código da versão é um número inteiro utilizado internamente pelo Google Play.
 - O Eclipse assina digitalmente seu aplicativo de forma automática para publicação no Google Play.
 - O Google atribui uma classificação de conteúdo ao seu aplicativo.

Esta página foi deixada em branco intencionalmente.

Introdução aos aplicativos Java



Objetivos

Neste capítulo, você vai:

- Escrever aplicativos Java simples.
- Usar instruções de entrada e saída.
- Conhecer os tipos primitivos da linguagem Java.
- Conhecer conceitos básicos sobre memória.
- Usar operadores aritméticos.
- Saber sobre a precedência dos operadores aritméticos.
- Escrever instruções de tomada de decisão.
- Usar operadores relacionais e de igualdade.



- | | |
|---|---|
| A.1 Introdução | A.6 Conceitos sobre memória |
| A.2 Seu primeiro programa em Java: impressão de uma linha de texto | A.7 Aritmética |
| A.3 Modificação de seu primeiro programa em Java | A.8 Tomada de decisão: operadores de igualdade e relacionais |
| A.4 Exibição de texto com <code>printf</code> | A.9 Para finalizar |
| A.5 Outro aplicativo: soma de valores inteiros | |

Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios

A.1 Introdução

Este apêndice apresenta a programação de aplicativos com Java. Você vai usar ferramentas do JDK para compilar e executar programas. Postamos um vídeo do Dive Into® (em inglês) no endereço www.deitel.com/books/AndroidHTP2/ para ajudá-lo a começar a usar o conhecido IDE (ambiente de desenvolvimento integrado) do Eclipse – o IDE Java mais amplamente utilizado e normalmente empregado para desenvolvimento de aplicativos Android.

A.2 Seu primeiro programa em Java: impressão de uma linha de texto

Um **aplicativo** Java é um programa de computador que é executado quando você usa o **comando java** para ativar a JVM (Java Virtual Machine). Primeiramente, vamos examinar um aplicativo simples que exibe uma linha de texto. A Figura A.1 mostra o programa e, em seguida, um quadro apresentando sua saída.

```

1 // Fig. A.1: Welcome1.java
2 // Programa para impressão de texto.
3
4 public class Welcome1
5 {
6     // o método main inicia a execução do aplicativo Java
7     public static void main( String[] args )
8     {
9         System.out.println( "Welcome to Java Programming!" );
10    } // fim do método main
11 } // fim da classe Welcome1

```

Welcome to Java Programming!

Figura A.1 Programa para impressão de texto.

Comentando seus programas

Inserimos **comentários** para **documentar os programas** e torná-los mais claros. O compilador Java ignora os comentários; portanto, eles *não* fazem o computador executar qualquer ação quando o programa é executado.

O comentário na linha 1

```
// Fig. A.1: Welcome1.java
```

começa com `//`, indicando que esse é um **comentário de fim de linha** – ele termina no final da linha na qual o `//` aparece. A linha 2 é um comentário que descreve a finalidade do programa.

A linguagem Java também tem **comentários tradicionais**, os quais podem abranger várias linhas, como em

```
/* Este é um comentário tradicional. Ele
   pode abranger várias linhas */
```

Eles começam e terminam com delimitadores, `/*` e `*/`. O compilador ignora todo o texto entre os delimitadores.



Erro de programação comum A.1

*Quando o compilador encontra código que viola as regras da linguagem Java (isto é, sua sintaxe), ocorre um **erro de sintaxe**. Os erros de sintaxe também são chamados de **erros de compilação**, pois o compilador os detecta durante a fase de compilação. O compilador responde emitindo uma mensagem de erro e impedindo que seu programa seja compilado.*

Usando linhas em branco

A linha 3 é uma linha em branco. Linhas em branco, caracteres de espaço e tabulações facilitam a leitura dos programas. Juntos, eles são conhecidos como **espaços em branco** (ou whitespace). O compilador ignora os espaços em branco.

Declarando uma classe

A linha 4 inicia a **declaração** da **classe** `Welcome1`. Todo programa Java consiste em pelo menos uma classe que você (o programador) define. A **palavra-chave** `class` introduz uma declaração de classe e é seguida imediatamente pelo **nome da classe** (`Welcome1`). **As palavras-chave** são reservadas para uso do Java e são sempre escritas apenas com letras minúsculas. A lista completa de palavras-chave pode ser vista no endereço:

<http://bit.ly/JavaKeywords>

Nomes de classe e identificadores

Por convenção, os nomes de classe começam com uma letra maiúscula, e a primeira letra de cada palavra que eles contêm é maiúscula (por exemplo, `ExemploDeNomeDeClasse`). Um nome de classe é um **identificador** – uma série de caracteres consistindo em letras, algarismos, sublinhados (`_`) e cifrões (`$`) que não começam com um algarismo e não contêm espaços. O nome `7button` não é um identificador válido porque começa com um algarismo, e o nome `input field` não é um identificador válido porque contém um espaço. A linguagem Java **diferencia letras maiúsculas e minúsculas**; portanto, `value` e `Value` são identificadores diferentes.

Nos apêndices de A a E, toda classe que definimos começa com a palavra-chave `public`. Para nosso aplicativo, o nome de arquivo é `Welcome1.java`.



Erro de programação comum A.2

Uma classe `public` deve ser colocada em um arquivo que tenha o mesmo nome da classe (tanto em termos de grafia como do uso de maiúsculas e minúsculas) mais a extensão `.java`; caso contrário, ocorrerá um erro de compilação. Por exemplo, a classe `public Welcome1` deve ser colocada em arquivo chamado `Welcome1.java`.

Uma **chave de abertura** (como na linha 5), `{`, inicia o **corpo** de toda declaração de classe. Uma **chave de fechamento** correspondente, `}`, deve terminar cada declaração de classe.



Boa prática de programação A.1

Recue o corpo inteiro de cada declaração de classe por um “nível” entre a chave de abertura e a chave de fechamento que delimitam o corpo da classe. Recomendamos usar três espaços para formar um nível de recuo. Esse formato destaca a estrutura da declaração de classe e facilita sua leitura.

Declarando um método

A linha 6 é um comentário de fim de linha indicando a finalidade das linhas 7 a 10 do programa. A linha 7 é o ponto de partida de todo aplicativo Java. Os **parênteses** após o identificador `main` indicam que esse é um elemento básico do programa, chamado **método**. Para um aplicativo Java, um dos métodos *precisa* se chamar `main` e deve ser definido como mostrado na linha 7. Os métodos executam tarefas e podem retornar informações ao terminarem suas tarefas. A palavra-chave **void** indica que esse método *não* vai retornar informação. Na linha 7, o código `String[] args` nos parênteses é uma parte obrigatória da declaração do método `main` – discutiremos isso no Apêndice E.

A chave de abertura na linha 8 inicia o **corpo da declaração de método**. Uma chave de fechamento correspondente deve terminá-lo (linha 10).

Apresentando saída com `System.out.println`

A linha 9 instrui o computador a executar uma ação – a saber, imprimir a **string** de caracteres contida entre as aspas duplas (mas não as aspas em si). Às vezes, uma string é chamada de **string de caracteres** ou **string literal**. Nas strings, os caracteres de espaço em branco *não* são ignorados pelo compilador. As strings não podem abranger várias linhas de código.

O objeto **System.out** é conhecido como **objeto de saída padrão**. Ele permite aos aplicativos Java exibir informações na **janela de comando** a partir da qual é executado. Nas versões recentes do Microsoft Windows, a janela de comando é o **Prompt de Comando**. No UNIX/Linux/Mac OS X, a janela de comando é chamada de **janela de terminal** ou **shell**. Muitos programadores a chamam simplesmente de **linha de comando**.

O método `System.out.println` exibe (ou imprime) uma linha de texto na janela de comando. A string entre parênteses na linha 9 é o **argumento** do método. Ao concluir sua tarefa, `System.out.println` posiciona o cursor (o local onde o próximo caractere será exibido) no início da próxima linha na janela de comando.

A linha 9 inteira, inclusive `System.out.println`, o argumento `"Welcome to Java Programming!"` nos parênteses e o **ponto e vírgula** (`;`), é chamada de **instrução**. A maioria das instruções termina com um ponto e vírgula. Quando a instrução da linha 9 é executada, ela exibe `Welcome to Java Programming!` na janela de comando.

Usando comentários de fim de linha em chaves de fechamento para aumentar a clareza

Incluímos um comentário de fim de linha após uma chave de fechamento que finaliza uma declaração de método e após uma chave de fechamento que finaliza uma declaração de classe. Por exemplo, a linha 10 indica a chave de fechamento do método `main`, e a linha 11 indica a chave de fechamento da classe `Welcome1`.

Compilando e executando seu primeiro aplicativo Java

Supomos que você está usando as ferramentas de linha de comando do Java Development Kit e não um IDE. Nosso Java Resource Centers, no endereço www.deitel.com/ResourceCenters.html, fornece links para tutoriais que o ajudam a começar a usar diversas ferramentas de desenvolvimento com Java populares, incluindo NetBeans™, Eclipse™

e outras. Também postamos um vídeo sobre Eclipse no endereço www.deitel.com/books/AndroidHTP2/, para ajudá-lo a começar a usar esse conhecido IDE.

A fim de se preparar para compilar o programa, abra uma janela de comando e mude para o diretório onde o programa está armazenado. Muitos sistemas operacionais usam o comando `cd` para mudar de diretório. No Windows, por exemplo,

```
cd c:\examples\appA\figA_01
```

muda para o diretório `figA_01`. No UNIX/Linux/Max OS X, o comando

```
cd ~/examples/appA/figA_01
```

muda para o diretório `figA_01`.

Para compilar o programa, digite

```
javac Welcome1.java
```

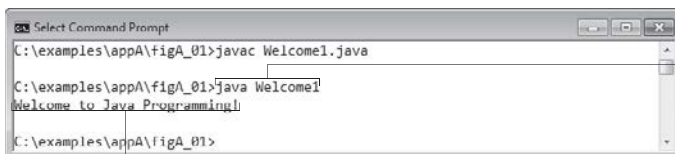
Se o programa não contém erros de sintaxe, esse comando cria um novo arquivo chamado `Welcome1.class` (conhecido como **arquivo de classe** de `Welcome1`), contendo o bytecode (código binário) em Java, independente de plataforma, que representa nosso aplicativo. Quando usarmos o comando `java` para executar o aplicativo em determinada plataforma, a JVM transformará esse bytecode em instruções compreendidas pelo sistema operacional e pelo hardware subjacentes.



Dica de prevenção de erro A.1

Ao tentar compilar um programa, se você receber uma mensagem como “bad command or filename”, “javac: command not found” ou “‘javac’ is not recognized as an internal or external command, operable program or batch file”, então sua instalação de software Java não foi concluída corretamente. Se estiver usando o JDK, isso indica que a variável de ambiente PATH do sistema não foi configurada corretamente. Examine cuidadosamente as instruções de instalação na seção “Antes de começar” deste livro. Em alguns sistemas, após corrigir a variável PATH, talvez seja preciso reinicializar o computador ou abrir uma nova janela de comando para que essas configurações entrem em vigor.

A Figura A.2 mostra o programa da Fig. A.1 sendo executado em uma janela **Prompt de Comando** do Microsoft® Windows® 7. Para executar o programa, digite `java Welcome1`. Esse comando ativa a JVM, a qual carrega o arquivo `.class` da classe `Welcome1`. O comando omite a extensão de nome de arquivo `.class`; caso contrário, a JVM não executará o programa. A JVM chama o método `main`. Em seguida, a instrução da linha 9 de `main` exibe “Welcome to Java Programming!”.



Você digita este comando para executar o aplicativo

O programa exibe a saída
Welcome to Java Programming! na tela

Figura A.2 Executando `Welcome1` a partir do **Prompt de Comando**.



Dica de prevenção de erro A.2

Ao tentar executar um programa Java, se você receber uma mensagem como “Exception in thread “main” java.lang.NoClassDefFoundError: Welcome1”, sua variável de ambiental CLASSPATH não foi configurada corretamente. Examine cuidadosamente as instruções de instalação na seção “Antes de começar” deste livro. Em alguns sistemas, talvez seja preciso reinicializar o computador ou abrir uma nova janela de comando após configurar a variável CLASSPATH.

A.3 Modificação de seu primeiro programa em Java

Welcome to Java Programming! pode ser exibido de diversas maneiras. A classe Welcome2, mostrada na Fig. A.3, usa duas instruções (linhas 9 e 10) para produzir a saída mostrada na Fig. A.1.

```

1 // Fig. A.3: Welcome2.java
2 // Imprimindo uma linha de texto com várias instruções.
3
4 public class Welcome2
5 {
6     // o método main inicia a execução do aplicativo Java
7     public static void main( String[] args )
8     {
9         System.out.print( "Welcome to " );
10        System.out.println( "Java Programming!" );
11    } // fim do método main
12 } // fim da classe Welcome2

```

Welcome to Java Programming!

Figura A.3 Imprimindo uma linha de texto com várias instruções.

O programa é semelhante ao da Fig. A.1; portanto, discutimos aqui somente as alterações. A linha 2 é um comentário declarando a finalidade do programa. A linha 4 inicia a declaração da classe Welcome2. As linhas 9 e 10 do método main exibem uma única linha de texto. A primeira instrução usa o método print de System.out para exibir uma string. Cada instrução print ou println retoma a exibição dos caracteres a partir de onde a última instrução print ou println parou de exibir caracteres. Ao contrário de println, depois de exibir seu argumento, print *não* posiciona o cursor de saída no início da próxima linha na janela de comando — o próximo caractere que o programa exibir vai aparecer *imediatamente após* o último caractere exibido por print. Assim, a linha 10 posiciona o primeiro caractere de seu argumento (a letra “J”) imediatamente após o último caractere exibido pela linha 9 (o *caractere de espaço* antes do caractere de aspas duplas de fechamento da string).

Exibindo várias linhas de texto com apenas uma instrução

Uma única instrução pode exibir várias linhas, usando **caracteres de nova linha**, os quais sinalizam aos métodos print e println de System.out quando posicionar o cursor de saída no início da próxima linha na janela de comando. Assim como as linhas em branco, caracteres de espaço e de tabulação, os caracteres de nova linha

são caracteres de espaço em branco. O programa da Fig. A.4 mostra quatro linhas de texto na saída usando caracteres de nova linha para determinar quando deve iniciar cada linha nova.

```

1 // Fig. A.4: Welcome3.java
2 // Imprimindo várias linhas de texto com apenas uma instrução.
3
4 public class Welcome3
5 {
6     // o método main inicia a execução do aplicativo Java
7     public static void main( String[] args )
8     {
9         System.out.println( "Welcome\nto\nJava\nProgramming!" );
10    } // fim do método main
11 } // fim da classe Welcome3

```

```

Welcome
to
Java
Programming!

```

Figura A.4 Imprimindo várias linhas de texto com apenas uma instrução.

A linha 2 é um comentário declarando a finalidade do programa. A linha 4 inicia a declaração da classe `Welcome3`. A linha 9 exibe quatro linhas de texto separadas na janela de comando. Normalmente, os caracteres de uma string são exibidos *exatamente* como aparecem nas aspas duplas. Note, entretanto, que os caracteres emparelhados `\n` (repetidos três vezes na instrução) não aparecem na tela. A **barra invertida** (`\`) é um **caractere de escape** que tem significado especial para os métodos `print` e `println` de `System.out`. Quando uma barra invertida aparece em uma string, a linguagem Java a combina com o caractere seguinte para formar uma **sequência de escape**. A sequência de escape `\n` representa o caractere de nova linha. Quando um caractere de nova linha aparece em uma string que está sendo gerada na saída por `System.out`, ele faz o cursor de saída da tela se mover para o início da próxima linha na janela de comando.

A Figura A.5 lista várias sequências de escape comuns e descreve como elas afetam a exibição de caracteres na janela de comando.

Sequência de escape	Descrição
<code>\n</code>	Nova linha. Posiciona o cursor da tela no início da próxima linha.
<code>\t</code>	Tabulação horizontal. Move o cursor da tela para a próxima parada de tabulação.
<code>\r</code>	Retorno de carro (carriage return). Posiciona o cursor da tela no início da linha atual – <i>não</i> avança para a próxima linha. Os caracteres gerados na saída após o retorno de carro sobrescrevem os caracteres produzidos anteriormente nessa linha.
<code>\\</code>	Barra invertida. Usada para imprimir um caractere de barra invertida.
<code>\"</code>	Aspas duplas. Usada para imprimir um caractere de aspas duplas. Por exemplo, <code>System.out.println("\"in quotes\"");</code> exibe <code>"in quotes"</code> .

Figura A.5 Algumas sequências de escape comuns.

A.4 Exibição de texto com printf

O método `System.out.printf` exibe dados formatados. A Figura A.6 usa esse método para produzir as strings "Welcome to" e "Java Programming!" na saída. As linhas 9 e 10 chamam o método `System.out.printf` para exibir a saída do programa. A chamada de método especifica três argumentos – eles são colocados em uma **lista separada por vírgulas**.

```

1 // Fig. A.6: Welcome4.java
2 // Exibindo várias linhas com o método System.out.printf.
3
4 public class Welcome4
5 {
6     // o método main inicia a execução do aplicativo Java
7     public static void main( String[] args )
8     {
9         System.out.printf( "%s\n%s\n",
10             "Welcome to", "Java Programming!" );
11     } // fim do método main
12 } // fim da classe Welcome4

```

```

Welcome to
Java Programming!

```

Figura A.6 Exibindo várias linhas com o método `System.out.printf`.

As linhas 9 e 10 representam apenas *uma* instrução. A linguagem Java permite que instruções grandes sejam divididas por muitas linhas. Recuamos a linha 10 para indicar que se trata de uma *continuação* da linha 9.

O primeiro argumento do método `printf` é uma **string de formato** que pode consistir em **texto fixo** e **especificadores de formato**. O texto fixo é mostrado na saída por `printf` exatamente como seria exibido por `print` ou `println`. Cada especificador de formato é um espaço reservado para um valor e define o tipo de dado a ser gerado na saída. Opcionalmente, os especificadores de formato também podem incluir informações de formatação.

Os especificadores de formato começam com um sinal de porcentagem (%), seguido de um caractere representando o tipo de dado. Por exemplo, o especificador de formato `%s` é um espaço reservado para uma string. A string de formato na linha 9 especifica que `printf` deve produzir duas strings na saída, cada uma seguida por um caractere de nova linha. Na posição do primeiro especificador de formato, `printf` substitui o valor do primeiro argumento após a string de formato. Na posição de cada especificador de formato subsequente, `printf` substitui o valor do próximo argumento. Assim, esse exemplo substitui "Welcome to" para o primeiro `%s` e "Java Programming!" para o segundo `%s`. A saída mostra que são exibidas duas linhas de texto.

A.5 Outro aplicativo: soma de valores inteiros

Nosso próximo aplicativo lê (ou insere) dois valores **inteiros** (números inteiros, como -22, 7, 0 e 1024) digitados pelo usuário no teclado, calcula a soma e a exibe. Os programas se lembram dos números e outros dados que estão na memória do computador e acessam esses dados por meio de elementos chamados **variáveis**. O programa da Fig. A.7 demonstra esses conceitos. No exemplo de saída, usamos texto em negrito para identificar a entrada do usuário (isto é, 45 e 72).

```

1 // Fig. A.7: Addition.java
2 // Programa de adição que exibe a soma de dois números.
3 import java.util.Scanner; // o programa usa a classe Scanner
4
5 public class Addition
6 {
7     // o método main inicia a execução do aplicativo Java
8     public static void main( String[] args )
9     {
10         // cria um Scanner para obter entrada da janela de comando
11         Scanner input = new Scanner( System.in );
12
13         int number1; // primeiro número a somar
14         int number2; // segundo número a somar
15         int sum; // soma de number1 e number2
16
17         System.out.print( "Enter first integer: " ); // prompt
18         number1 = input.nextInt(); // lê o primeiro número do usuário
19
20         System.out.print( "Enter second integer: " ); // prompt
21         number2 = input.nextInt(); // lê o segundo número do usuário
22
23         sum = number1 + number2; // soma os números e armazena o total em sum
24
25         System.out.printf( "Sum is %d\n", sum ); // exibe sum
26     } // fim do método main
27 } // fim da classe Addition

```

```

Enter first integer: 45
Enter second integer: 72
Sum is 117

```

Figura A.7 Programa de adição que exibe a soma de dois números.

Declarações *import*

As linhas 1 e 2 declaram o número da figura, o nome do arquivo e a finalidade do programa. Uma importante vantagem da linguagem Java é seu rico conjunto de classes predefinidas que você pode *reutilizar* em vez de “reinventar a roda”. Essas classes são agrupadas em **pacotes** – grupos nomeados de classes relacionadas – e, coletivamente, são denominadas **biblioteca de classes Java** ou **API Java (Interface de Programação de Aplicativos Java)**. A linha 3 é uma **declaração import** que ajuda o compilador a localizar uma classe utilizada nesse programa. Ela indica que esse exemplo utiliza a classe predefinida `Scanner` (discutida em breve) do pacote `java.util` da linguagem Java.

Declarando a classe *Addition*

A linha 5 inicia a declaração da classe `Addition`. O nome de arquivo dessa classe `public` deve ser `Addition.java`. Lembre-se de que o corpo de cada declaração de classe começa com uma chave de abertura (linha 6) e termina com uma chave de fechamento (linha 27).

O aplicativo começa a executar com o método `main` (linhas 8 a 26). A chave de abertura (linha 9) marca o início do corpo do método `main`, e a chave de fechamento correspondente (linha 26) marca seu fim. O método `main` está recuado por um nível no corpo da classe `Addition` e, por clareza, o código do corpo de `main` está recuado outro nível.

Declarando e criando um objeto Scanner para obter entrada do usuário a partir do teclado

Uma **variável** é um lugar na memória do computador onde se pode armazenar um valor para uso posterior em um programa. Todas as variáveis Java *devem* ser declaradas com um **nome** e um **tipo** *antes* de serem usadas. O nome de uma variável permite que o programa acesse seu valor na memória. O nome de uma variável pode ser qualquer identificador válido. O tipo de uma variável especifica o tipo de informação armazenada nesse local da memória. Como as outras instruções, as instruções de declaração terminam com um ponto e vírgula (;).

A linha 11 é uma **instrução de declaração de variável** que especifica o nome (input) e o tipo (Scanner) de uma variável que é usada nesse programa. Um objeto **Scanner** permite que um programa leia dados (por exemplo, números e strings) para uso em um aplicativo. Os dados podem ser provenientes de muitas fontes, como de um usuário ao teclado ou de um arquivo no disco. Antes de usar um objeto Scanner, você precisa criá-lo e especificar a fonte dos dados.

O sinal = na linha 11 indica que, em sua declaração, a variável Scanner input deve ser **inicializada** (isto é, preparada para uso no programa) com o resultado da expressão à direita do sinal de igual – new Scanner(System.in). Essa expressão utiliza a palavra-chave **new** para criar um objeto Scanner que lê os caracteres digitados no teclado pelo usuário. O **objeto de entrada padrão, System.in**, permite que os aplicativos leiam os bytes de informação digitados pelo usuário. O objeto Scanner transforma esses bytes em tipos (como, por exemplo, valores int) que podem ser usados em um programa.

Declarando variáveis para armazenar valores inteiros

As instruções de declaração de variável nas linhas 13 a 15 declaram que as variáveis number1, number2 e sum armazenam dados de tipo **int** – ou seja, valores inteiros (números inteiros, como 72, -1127 e 0). Essas variáveis ainda não foram inicializadas. O intervalo de valores para uma variável int é de -2.147.483.648 a +2.147.483.647. [Obs.: os valores int reais não podem conter vírgulas.]

Outros tipos de dados incluem **float** e **double**, para armazenar números reais (como 3,4; 0,0 e -11,19), e **char**, para armazenar dados de caractere. As variáveis de tipo char representam caracteres individuais, como uma letra maiúscula (por exemplo, A), um algarismo (por exemplo, 7), um caractere especial (por exemplo, * ou %) ou uma sequência de escape (por exemplo, o caractere de nova linha, \n). Os tipos int, float, double e char são denominados **tipos primitivos**. Os nomes de tipo primitivo são palavras-chave e devem aparecer com todas as letras minúsculas. O Apêndice L resume as características dos oito tipos primitivos (boolean, byte, char, short, int, long, float e double).



Boa prática de programação A.2

Por convenção, os identificadores de nome de variável começam com uma letra minúscula e toda palavra no nome, após a primeira, começa com uma letra maiúscula.

Solicitando entrada do usuário

A linha 17 usa System.out.print para exibir a mensagem "Enter first integer: ". Essa mensagem é chamada de **prompt** porque instrui o usuário a tomar uma iniciativa específica. Usamos o método print aqui, em vez de println, para que a entrada do usuário apareça na mesma linha do prompt. Lembre que na Seção A.2 os identificadores que começam com letras maiúsculas normalmente representam nomes de classe. Portanto,

System é uma classe. A classe System faz parte do pacote `java.lang`. Ela não é importada com uma declaração `import` no início do programa.



Observação sobre engenharia de software A.1

Por padrão, o pacote `java.lang` é importado em todo programa Java; assim, as classes em `java.lang` são as únicas na API Java que não exigem uma declaração `import`.

Obtendo um valor `int` como entrada do usuário

A linha 18 usa o método `nextInt` do objeto Scanner `input` para obter um valor inteiro do usuário no teclado. Nesse ponto, o programa espera que o usuário digite o número e pressione a tecla `Enter` para enviá-lo para o programa.

Nosso programa presume que o usuário digita um valor inteiro válido. Caso contrário, ocorrerá um erro de lógica no tempo de execução (runtime) e o programa terminará. O Apêndice H discute como tornar seus programas mais robustos, permitindo que eles tratem tais erros – isso torna seu programa mais *tolerante à falha*.

Na linha 18, colocamos o resultado da chamada ao método `nextInt` (um valor `int`) na variável `number1` usando o **operador de atribuição**, `=`. A instrução é lida como “`number1` recebe o valor de `input.nextInt()`”. O operador `=` é chamado de **operador binário**, pois tem dois **operandos** – `number1` e o resultado da chamada de método `input.nextInt()`. Essa instrução é chamada de instrução de atribuição, pois atribui um valor a uma variável. Tudo que está à *direita* do operador de atribuição, `=`, é sempre avaliado *antes* que a atribuição seja realizada.



Boa prática de programação A.3

Colocar espaços nos dois lados de um operador binário torna o programa mais legível.

Solicitando e inserindo um segundo valor `int`

A linha 20 pede para o usuário digitar o segundo valor inteiro. A linha 21 lê o segundo valor inteiro e o atribui à variável `number2`.

Usando variáveis em um cálculo

A linha 23 é uma instrução de atribuição que calcula a soma das variáveis `number1` e `number2` e, então, atribui o resultado à variável `sum` usando o operador de atribuição, `=`. A instrução é lida como “`sum` recebe o valor de `number1 + number2`”. Em geral, cálculos são efetuados em instruções de atribuição. Quando o programa encontra a operação de adição, efetua o cálculo usando os valores armazenados nas variáveis `number1` e `number2`. Na instrução anterior, o operador de adição é *binário* – seus *dois* operandos são as variáveis `number1` e `number2`. As partes das instruções que contêm cálculos são denominadas **expressões**. Na verdade, uma expressão é qualquer parte de uma instrução que tem um *valor* associado. Por exemplo, o valor da expressão `number1 + number2` é a *soma* dos números. Do mesmo modo, o valor da expressão `input.nextInt()` é o inteiro digitado pelo usuário.

Exibindo o resultado do cálculo

Após o cálculo ser efetuado, a linha 25 utiliza o método `System.out.printf` para exibir `sum`. O especificador de formato `%d` é um espaço reservado para um valor `int` (neste caso, o valor de `sum`) – a letra `d` significa “inteiro decimal”. Os caracteres restantes na string de formato são todo o texto fixo. Assim, o método `printf` exibe “Sum is ”, seguido pelo

valor de `sum` (na posição do especificador de formato `%d`) e um caractere de nova linha. Os cálculos também podem ser efetuados *dentro* de instruções `printf`. Poderíamos ter combinado as instruções das linhas 23 e 25 na instrução

```
System.out.printf( "Sum is %d\n", ( number1 + number2 ) );
```

Os parênteses em torno da expressão `number1 + number2` não são obrigatórios – eles foram incluídos para realçar o fato de que toda a expressão é gerada na saída, na posição do especificador de formato `%d`.

Documentação da API Java

Para cada nova classe da API Java que utilizamos, indicamos o pacote em que está localizada. Essa informação o ajuda a localizar descrições de cada pacote e classe na documentação da API Java. Uma versão baseada na web dessa documentação pode ser encontrada em

```
docs.oracle.com/javase/6/docs/api/
```

O download pode ser feito no endereço

```
www.oracle.com/technetwork/java/javase/downloads/index.html
```

A.6 Conceitos sobre memória

Nomes de variável como `number1`, `number2` e `sum` na verdade correspondem a locais na memória do computador. Toda variável tem um **nome**, um **tipo**, um **tamanho** (em bytes) e um **valor**. No programa de adição da Fig. A.7, quando a seguinte instrução (linha 18) é executada:

```
number1 = input.nextInt(); // lê o primeiro número do usuário
```

o número digitado pelo usuário é colocado em um lugar na memória correspondente ao nome `number1`. Suponha que o usuário digite 45. O computador coloca esse valor inteiro em `number1` (Fig. A.8), substituindo o valor anterior (se houver) que estava nesse local. O valor anterior é perdido.

number1	45
---------	----

Figura A.8 Lugar na memória mostrando o nome e o valor da variável `number1`.

Quando a instrução (linha 21)

```
number2 = input.nextInt(); // lê o segundo número do usuário
```

é executada, suponha que o usuário digite 72. O computador coloca esse valor inteiro no local `number2`. Agora a memória aparece como mostrado na Figura A.9.

number1	45
number2	72

Figura A.9 Locais na memória depois de armazenar valores para `number1` e `number2`.

Depois que o programa da Fig. A.7 obtém valores para `number1` e `number2`, ele soma os valores e coloca o total na variável `sum`. A instrução (linha 23)

```
sum = number1 + number2; // soma os números e armazena o total em sum
```

efetua a soma e, então, substitui qualquer valor anterior que estava em `sum`. Após `sum` ter sido calculada, a memória aparece como na Fig. A.10. `number1` e `number2` contêm os valores que foram usados no cálculo de `sum`. Quando o cálculo foi efetuado, esses valores foram usados, mas não destruídos. Quando um valor é lido de um local da memória, o processo é não destrutivo.

number1	45
number2	72
sum	117

Figura A.10 Locais na memória depois de armazenar a soma de `number1` e `number2`.

A.7 Aritmética

A maioria dos programas efetua cálculos aritméticos. Os **operadores aritméticos** estão resumidos na Fig. A.11. Observe o uso de vários símbolos especiais não empregados na álgebra. O **asterisco** (`*`) indica multiplicação, e o sinal de porcentagem (`%`) é o **operador resto**, o qual vamos discutir em breve. Os operadores aritméticos da Fig. A.11 são *binários*, pois cada um deles trabalha com *dois* operandos. Por exemplo, a expressão `f + 7` contém o operador binário `+` e dois operandos, `f` e `7`.

Operação em Java	Operador	Expressão algébrica	Expressão em Java
Adição	<code>+</code>	$f + 7$	<code>f + 7</code>
Subtração	<code>-</code>	$p - c$	<code>p - c</code>
Multiplicação	<code>*</code>	bm	<code>b * m</code>
Divisão	<code>/</code>	x/y ou $\frac{x}{y}$ ou $x \div y$	<code>x / y</code>
Resto	<code>%</code>	$r \bmod s$	<code>r % s</code>

Figura A.11 Operadores aritméticos.

A **divisão inteira** gera um quociente inteiro. Por exemplo, a expressão `7/4` é avaliada como `1`, e a expressão `17/5` é avaliada como `3`. Qualquer parte fracionária na divisão inteira é simplesmente *descartada* (isto é, *truncada*) – não ocorre arredondamento. A linguagem Java fornece o operador resto, `%`, o qual gera o resto após a divisão. A expressão `x%y` gera o resto após `x` ser dividido por `y`. Assim, `7%4` produz `3`, e `17%5` produz `2`. O uso mais comum desse operador se dá com operandos inteiros, mas também pode ser usado com outros tipos aritméticos.

Expressões aritméticas em linha reta

Em Java, as expressões aritméticas devem ser escritas **em linha reta** (em uma só linha) para facilitar a digitação dos programas no computador. Assim, expressões como “a di-

vidido por b” devem ser escritas como a/b, de modo que todas as constantes, variáveis e operadores apareçam em uma linha reta. Os compiladores geralmente não aceitam a seguinte notação algébrica:

$$\frac{a}{b}$$

Parênteses para agrupar subexpressões

Parênteses são usados para agrupar termos em expressões Java da mesma maneira que nas expressões algébricas. Por exemplo, para multiplicar a pelo valor de b + c, escrevemos

```
a * ( b + c )
```

Se uma expressão contém **parênteses aninhados**, como em

```
( ( a + b ) * c )
```

a expressão no conjunto interno de parênteses (neste caso, a + b) é avaliada primeiro.

Regras de precedência de operador

A linguagem Java aplica os operadores de expressões aritméticas em uma sequência precisa, determinada pelas **regras de precedência de operador**, as quais de um modo geral são as mesmas seguidas na álgebra:

1. As operações de multiplicação, divisão e resto são aplicadas primeiro. Se uma expressão contém várias dessas operações, elas são aplicadas da esquerda para a direita. Os operadores de multiplicação, divisão e resto têm o mesmo nível de precedência.
2. As operações de adição e subtração são aplicadas em seguida. Se uma expressão contém várias dessas operações, os operadores são aplicados da esquerda para a direita. Os operadores de adição e subtração têm o mesmo nível de precedência.

Essas regras permitem à linguagem Java aplicar os operadores na ordem correta.¹ Quando dizemos que os operadores são aplicados da esquerda para a direita, estamos nos referindo à sua **associatividade**. Alguns operadores são associados da direita para a esquerda. A Figura A.12 resume essas regras de precedência de operador. Há uma tabela de precedência completa no Apêndice K.

Operador(es)	Operação(ões)	Ordem de avaliação (precedência)
*	Multiplicação	Avaliado primeiro. Se houver vários operadores desse tipo, eles serão avaliados da esquerda para a direita.
/	Divisão	
%	Resto	
+	Adição	Avaliado em seguida. Se houver vários operadores desse tipo, eles serão avaliados da esquerda para a direita.
-	Subtração	
=	Atribuição	Avaliado por último.

Figura A.12 Precedência dos operadores aritméticos.

¹ Usamos exemplos simples para explicar a ordem de avaliação de expressões. Nas expressões mais complexas que você vai encontrar, ocorrerão problemas sutis. Para obter mais informações sobre a ordem de avaliação, consulte o Capítulo 15 de *The Java™ Language Specification* (java.sun.com/docs/books/jls/).

Exemplos de expressões algébricas e de expressões em Java

Agora, vamos examinar várias expressões considerando as regras de precedência de operador. Cada exemplo lista uma expressão algébrica e sua expressão equivalente em Java. A seguir, está um exemplo de média aritmética de cinco termos:

$$\begin{array}{ll} \text{Álgebra:} & m = \frac{a + b + c + d + e}{5} \\ \text{Java:} & m = (a + b + c + d + e) / 5; \end{array}$$

Os parênteses são exigidos porque a divisão tem precedência mais alta que a adição. A quantidade inteira $(a + b + c + d + e)$ deve ser dividida por 5. Se os parênteses forem erroneamente omitidos, obteremos $a + b + c + d + e / 5$, que é avaliado como

$$a + b + c + d + \frac{e}{5}$$

Aqui está um exemplo da equação em linha reta:

$$\begin{array}{ll} \text{Álgebra:} & y = mx + b \\ \text{Java:} & y = m * x + b; \end{array}$$

Nenhum parêntese é exigido. O operador de multiplicação é aplicado primeiro porque a multiplicação tem precedência mais alta que a adição. A atribuição ocorre por último, pois tem precedência mais baixa que a multiplicação ou adição.

O exemplo a seguir contém operações de resto (%), multiplicação, divisão, adição e subtração:

$$\begin{array}{ll} \text{Álgebra:} & z = pr \% q + w/x - y \\ \text{Java:} & z = p * r \% q + w / x - y; \end{array}$$

6
1
2
4
3
5

Os números circulados sob a instrução indicam a ordem na qual a linguagem Java aplica os operadores. As operações *, % e / são avaliadas primeiro, da esquerda para a direita (isto é, elas são associadas da esquerda para a direita), pois têm precedência mais alta que + e -. As operações + e - são avaliadas em seguida. Essas operações também são aplicadas da esquerda para a direita. A operação de atribuição (=) é avaliada por último.

Avaliação de um polinômio do segundo grau

Para entender melhor as regras de precedência de operador, considere a avaliação de uma expressão de atribuição que inclui um polinômio do segundo grau $ax^2 + bx + c$:

$$y = a * x * x + b * x + c;$$

6
1
2
4
3
5

As operações de multiplicação são avaliadas primeiro, da esquerda para a direita (isto é, elas são associadas da esquerda para a direita), pois têm precedência mais alta que a adição. (A linguagem Java não tem operador aritmético para exponenciação; portanto x^2 é representado como $x * x$. A seção C.16 mostra um modo alternativo de fazer exponenciação.) As operações de adição são avaliadas em seguida, da esquerda para a direita. Suponha que a, b, c e x sejam inicializadas (recebam valores) como segue: $a = 2$, $b = 3$, $c = 7$ e $x = 5$. A Figura A.13 ilustra a ordem em que os operadores são aplicados.

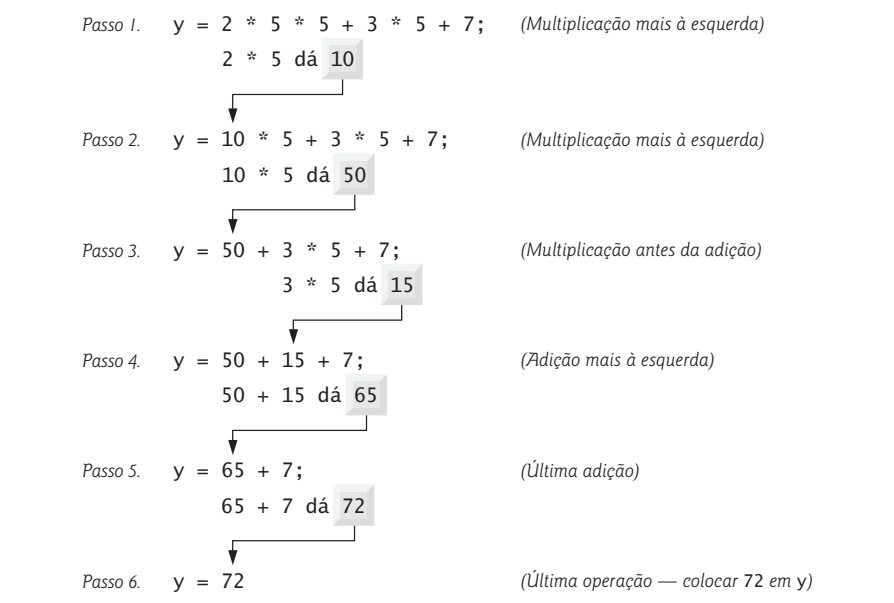


Figura A.13 | Ordem na qual um polinômio de segundo grau é avaliado.

A.8 Tomada de decisão: operadores de igualdade e relacionais

Uma **condição** é uma expressão que pode ser **verdadeira** ou **falsa**. Esta seção apresenta a **instrução de seleção** `if` da linguagem Java, a qual permite a um programa tomar uma **decisão** com base no valor de uma condição. Por exemplo, a condição “nota maior ou igual a 60” determina se um aluno passou em um teste. Se a condição de uma instrução `if` é verdadeira, o corpo da instrução é executado. Se a condição é falsa, o corpo não é executado. Vamos ver um exemplo em breve.

Nas instruções `if`, as condições podem ser formadas com os **operadores de igualdade** (`==` e `!=`) e **relacionais** (`>`, `<`, `>=` e `<=`) resumidos na Fig. A.14. Os dois operadores de igualdade têm o mesmo nível de precedência, o qual é *mais baixo* que o dos operadores relacionais. Os operadores de igualdade são associados da esquerda para a direita. Os operadores relacionais têm o mesmo nível de precedência e também são associados da esquerda para a direita.

Operador de igualdade ou relacional algébrico padrão	Operador de igualdade ou relacional em Java	Exemplo de condição em Java	Significado da condição em Java
Operadores de igualdade			
<code>=</code>	<code>==</code>	<code>x == y</code>	<code>x</code> é igual a <code>y</code>
<code>≠</code>	<code>!=</code>	<code>x != y</code>	<code>x</code> não é igual a <code>y</code>

Figura A.14 Operadores de igualdade e relacionais. (continua)

Operador de igualdade ou relacional algébrico padrão	Operador de igualdade ou relacional em Java	Exemplo de condição em Java	Significado da condição em Java
Operadores relacionais			
>	>	x > y	x é maior que y
<	<	x < y	x é menor que y
≥	>=	x >= y	x é maior ou igual a y
≤	<=	x <= y	x é menor ou igual a y

Figura A.14 Operadores de igualdade e relacionais.

A Figura A.15 usa seis instruções `if` para comparar dois valores inteiros digitados pelo usuário. Se a condição de qualquer uma dessas instruções `if` é verdadeira, a instrução associada a essa instrução `if` é executada; caso contrário, a instrução é pulada. Usamos `Scanner` para inserir os valores inteiros do usuário e os armazenamos nas variáveis `number1` e `number2`. O programa compara os números e mostra os resultados das comparações que são verdadeiras.

```

1 // Fig. A.15: Comparison.java
2 // Compara valores inteiros usando instruções if,
3 // operadores relacionais e operadores de igualdade.
4 import java.util.Scanner; // o programa usa a classe Scanner
5
6 public class Comparison
7 {
8     // o método main inicia a execução do aplicativo Java
9     public static void main( String[] args )
10    {
11        // cria Scanner para obter entrada da linha de comando
12        Scanner input = new Scanner( System.in );
13
14        int number1; // primeiro número a comparar
15        int number2; // segundo número a comparar
16
17        System.out.print( "Enter first integer: " ); // prompt
18        number1 = input.nextInt(); // lê o primeiro número do usuário
19
20        System.out.print( "Enter second integer: " ); // prompt
21        number2 = input.nextInt(); // lê o segundo número do usuário
22
23        if ( number1 == number2 )
24            System.out.printf( "%d == %d\n", number1, number2 );
25
26        if ( number1 != number2 )
27            System.out.printf( "%d != %d\n", number1, number2 );
28
29        if ( number1 < number2 )
30            System.out.printf( "%d < %d\n", number1, number2 );
31
32        if ( number1 > number2 )
33            System.out.printf( "%d > %d\n", number1, number2 );

```

Figura A.15 Compara valores inteiros usando instruções `if`, operadores relacionais e operadores de igualdade. (continua)

```

34
35     if ( number1 <= number2 )
36         System.out.printf( "%d <= %d\n", number1, number2 );
37
38     if ( number1 >= number2 )
39         System.out.printf( "%d >= %d\n", number1, number2 );
40 } // fim do método main
41 } // fim da classe Comparison

```

```

Enter first integer: 777
Enter second integer: 777
777 == 777
777 <= 777
777 >= 777

```

```

Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000

```

```

Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000

```

Figura A.15 Compara valores inteiros usando instruções `if`, operadores relacionais e operadores de igualdade.

A declaração da classe `Comparison` começa na linha 6. O método `main` da classe (linhas 9 a 40) inicia a execução do programa. A linha 12 declara a variável `Scanner input` e atribui a ela um objeto `Scanner` que insere os dados da entrada padrão (isto é, o teclado).

As linhas 14 e 15 declaram as variáveis `int` utilizadas para armazenar os valores digitados pelo usuário.

As linhas 17 e 18 pedem para o usuário digitar e inserir o primeiro valor inteiro, respectivamente. O valor digitado é armazenado na variável `number1`.

As linhas 20 e 21 pedem para o usuário digitar e inserir o segundo valor inteiro, respectivamente. O valor digitado é armazenado na variável `number2`.

As linhas 23 e 24 comparam os valores de `number1` e `number2` para determinar se são iguais. Uma instrução `if` sempre começa com a palavra-chave `if`, seguida por uma condição entre parênteses. Uma instrução `if` espera uma instrução em seu corpo, mas pode conter várias, caso sejam incluídas em um conjunto de chaves `{ }`. O recuo da instrução do corpo mostrado aqui não é obrigatório, mas melhora a legibilidade do programa, enfatizando que a instrução da linha 24 *faz parte da* instrução `if` que começa na linha 23. A linha 24 só é executada se os números armazenados nas variáveis `number1` e `number2` são iguais (isto é, se a condição é verdadeira). As instruções `if` das linhas 26 e 27, 29 e 30,

32 e 33, 35 e 36, e 38 e 39 comparam `number1` e `number2` usando os operadores `!=`, `<`, `>`, `<=` e `>=`, respectivamente. Se a condição de uma ou mais das instruções `if` é verdadeira, a instrução do corpo correspondente é executada.



Erro de programação comum A.3

Confundir o operador de igualdade, `==`, com o operador de atribuição, `=`, pode causar um erro de lógica ou de sintaxe. O operador de igualdade deve ser lido como “é igual a”, e o operador de atribuição como “recebe” ou “recebe o valor de”. Para evitar confusão, algumas pessoas leem o operador de igualdade como “duplo igual” ou “igual igual”.

Não há ponto e vírgula (;) no final da primeira linha de cada instrução `if`. Tal ponto e vírgula resultaria em um erro de lógica em tempo de execução. Por exemplo,

```
if ( number1 == number2 ); // erro de lógica
    System.out.printf( "%d == %d\n", number1, number2 );
```

seria interpretado pelo Java como

```
if ( number1 == number2 )
; // instrução vazia
    System.out.printf( "%d == %d\n", number1, number2 );
```

onde o ponto e vírgula sozinho na linha – o que é chamado de **instrução vazia** – é a instrução a ser executada se a condição da instrução `if` for verdadeira. Quando a instrução vazia é executada, nenhuma tarefa é realizada. Então, o programa continua com a instrução de saída, a qual é sempre executada, independentemente de a condição ser verdadeira ou falsa, pois não faz parte da instrução `if`.

Observe o uso de espaço em branco na Fig. A.15. Lembre-se de que o compilador normalmente ignora espaço em branco. Assim, as instruções podem ser divididas por várias linhas e serem espaçadas de acordo com suas preferências, sem afetar o significado do programa. É incorreto dividir identificadores e strings. De preferência, deve-se manter as instruções pequenas, mas nem sempre isso é possível.

A Figura A.16 mostra os operadores discutidos até aqui, em ordem decrescente de precedência. Todos eles, menos o operador de atribuição, `=`, são associados da esquerda para a direita. O operador de atribuição, `=`, é associado da direita para a esquerda; portanto, uma expressão como `x = y = 0` é avaliada como se tivesse sido escrita como `x = (y = 0)`, a qual primeiro atribui o valor 0 à variável `y` e depois atribui o resultado dessa atribuição, 0, a `x`.

Operadores	Associatividade	Tipo
* / %	esquerda para a direita	multiplicativos
+ -	esquerda para a direita	aditivos
< <= > >=	esquerda para a direita	relacionais
== !=	esquerda para a direita	igualdade
=	direita para a esquerda	atribuição

Figura A.16 Precedência e associatividade dos operadores discutidos.

A.9 Para finalizar

Neste apêndice, você conheceu muitos recursos importantes da linguagem Java, incluindo a exibição de dados na tela em um **Prompt de Comando**, a inserção de dados a partir do teclado, como efetuar cálculos e como tomar decisões. Os aplicativos apresentados aqui mostraram os conceitos básicos de programação. Conforme você vai ver no Apêndice B, normalmente os aplicativos Java contêm apenas algumas linhas de código no método `main` – de modo geral, essas instruções criam os objetos que fazem o trabalho do aplicativo. No Apêndice B, você vai aprender a implementar suas próprias classes e a usar objetos dessas classes em aplicativos.

Exercícios de revisão

- A.1** Preencha os espaços em branco em cada um dos seguintes enunciados:
- Um(a) _____ inicia o corpo de todo método, e um(a) _____ finaliza o corpo de todo método.
 - A instrução _____ é usada para tomar decisões.
 - _____ inicia um comentário de fim de linha.
 - _____, _____ e _____ são chamados de espaços em branco.
 - _____ são reservadas para uso pela linguagem Java.
 - Os aplicativos Java começam a executar no método _____.
 - Os métodos _____, _____ e _____ exibem informações em uma janela de comando.
- A.2** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- Os comentários fazem o computador imprimir na tela o texto após o `//` quando o programa é executado.
 - Todas as variáveis devem receber um tipo ao serem declaradas.
 - A linguagem Java considera as variáveis `number` e `NUMBER` como sendo idênticas.
 - O operador resto (%) só pode ser usado com operandos inteiros.
 - Os operadores aritméticos `*`, `/`, `%`, `+` e `-` têm todos eles o mesmo nível de precedência.
- A.3** Escreva instruções para realizar cada uma das tarefas a seguir:
- Declarar as variáveis `c`, `thisIsAVariable`, `q76354` e `number` como sendo de tipo `int`.
 - Pedir ao usuário para que digite um valor inteiro.
 - Inserir um valor inteiro e atribuir o resultado à variável `int value`. Suponha que a variável `Scanner input` possa ser usada para ler um valor do teclado.
 - Imprimir "This is a Java program" em uma única linha na janela de comando. Use o método `System.out.println`.
 - Imprimir "This is a Java program" em duas linhas na janela de comando. A primeira linha deve terminar com `Java`. Use o método `System.out.println`.
 - Imprimir "This is a Java program" em duas linhas na janela de comando. A primeira linha deve terminar com `Java`. Use o método `System.out.printf` e dois especificadores de formato `%s`.
 - Se a variável `number` não for igual a 7, exiba "The variable number is not equal to 7".
- A.4** Identifique e corrija os erros em cada uma das seguintes instruções:
- ```
if (c < 7);
 System.out.println("c is less than 7");
```
  - ```
if ( c => 7 )
    System.out.println( "c is equal to or greater than 7" );
```

- A.5** Escreva declarações, instruções ou comentários que executem cada uma das seguintes tarefas:
- Declarar que um programa vai calcular o produto de três valores inteiros.
 - Criar um objeto `Scanner` chamado `input` que leia valores da entrada padrão.
 - Declarar as variáveis `x`, `y`, `z` e `result` como sendo de tipo `int`.
 - Pedir ao usuário para que digite o primeiro valor inteiro.
 - Ler o primeiro valor inteiro do usuário e armazená-lo na variável `x`.
 - Pedir ao usuário para que digite o segundo valor inteiro.
 - Ler o segundo valor inteiro do usuário e armazená-lo na variável `y`.
 - Pedir ao usuário para que digite o terceiro valor inteiro.
 - Ler o terceiro valor inteiro do usuário e armazená-lo na variável `z`.
 - Calcular o produto dos três valores inteiros contidos nas variáveis `x`, `y` e `z`, e atribuir o resultado à variável `result`.
 - Exibir a mensagem "Product is", seguida do valor da variável `result`.
- A.6** Usando as instruções que você escreveu no Exercício A.5, escreva um programa completo que calcule e imprima o produto de três valores inteiros.

Respostas dos exercícios de revisão

- A.1** a) chave de abertura (`{`), chave de fechamento (`}`). b) `if`. c) `//`. d) Caracteres de espaço, novas linhas e tabulações. e) Palavras-chave. f) `main`. g) `System.out.print`, `System.out.println` e `System.out.printf`.
- A.2** a) Falsa. Os comentários não fazem uma ação ser realizada quando o programa é executado. Eles são usados para documentar programas e torná-los mais claros. b) Verdadeira. c) Falsa. A linguagem Java diferencia letras maiúsculas e minúsculas; portanto, essas variáveis são distintas. d) Falsa. Em Java, o operador resto também pode ser usado com operandos que não são valores inteiros. e) Falsa. Os operadores `*`, `/` e `%` têm precedência mais alta que os operadores `+` e `-`.
- A.3** a) `int c, thisIsAVariable, q76354, number;`
ou
`int c;`
`int thisIsAVariable;`
`int q76354;`
`int number;`
b) `System.out.print( "Enter an integer: " );`
c) `value = input.nextInt();`
d) `System.out.println( "This is a Java program" );`
e) `System.out.println( "This is a Java\nprogram" );`
f) `System.out.printf( "%s\n%s\n", "This is a Java", "program" );`
g) `if ( number != 7 )`
`System.out.println( "The variable number is not equal to 7" );`
- A.4** a) Erro: ponto e vírgula após o parêntese da direita da condição (`c < 7`) no `if`. Correção: remover o ponto e vírgula após o parêntese da direita. [*Obs.*: como resultado, a instrução de saída vai ser executada independentemente de a condição no `if` ser verdadeira.]
b) Erro: O operador relacional `=>` está incorreto.
Correção: Alterar `=>` para `>=`.
- A.5** a) `// Calcula o produto de três valores inteiros`
b) `Scanner input = new Scanner( System.in );`

```

c) int x, y, z, result;
   ou
   int x;
   int y;
   int z;
   int result;
d) System.out.print( "Enter first integer: " );
e) x = input.nextInt();
f) System.out.print( "Enter second integer: " );
g) y = input.nextInt();
h) System.out.print( "Enter third integer: " );
i) z = input.nextInt();
j) result = x * y * z;
k) System.out.printf( "Product is %d\n", result );

```

A.6 A solução do Exercício de Revisão A.6 é a seguinte:

```

1 // Ex. 2.6: Product.java
2 // Calcula o produto de três valores inteiros
3 import java.util.Scanner; // o programa usa Scanner
4
5 public class Product
6 {
7     public static void main( String[] args )
8     {
9         // cria um objeto Scanner para obter entrada da janela de comando
10        Scanner input = new Scanner( System.in );
11
12        int x; // primeiro número digitado pelo usuário
13        int y; // segundo número digitado pelo usuário
14        int z; // terceiro número digitado pelo usuário
15        int result; // produto dos números
16
17        System.out.print( "Enter first integer: " ); // solicita a entrada
18        x = input.nextInt(); // lê o primeiro valor inteiro
19
20        System.out.print( "Enter second integer: " ); // solicita a entrada
21        y = input.nextInt(); // lê o segundo valor inteiro
22
23        System.out.print( "Enter third integer: " ); // solicita a entrada
24        z = input.nextInt(); // lê o terceiro valor inteiro
25
26        result = x * y * z; // calcula o produto dos números
27
28        System.out.printf( "Product is %d\n", result );
29    } // fim do método main
30 } // fim da classe Product

```

```

Enter first integer: 10
Enter second integer: 20
Enter third integer: 30
Product is 6000

```

Exercícios

A.7 Preencha os espaços em branco em cada um dos seguintes enunciados:

- _____ são usados para documentar um programa e torná-lo mais claro.
- Em um programa Java, uma decisão pode ser tomada com um(a) _____.
- Em geral, cálculos são efetuados em instruções _____.
- Os operadores aritméticos com a mesma precedência da multiplicação são _____ e _____.

- e) Quando parênteses em uma expressão aritmética são aninhados, o conjunto de parênteses _____ é avaliado primeiro.
- f) Um local na memória do computador que pode conter diferentes valores em vários momentos ao longo da execução de um programa é denominado _____.

A.8 Escreva instruções em Java que realizem cada uma das tarefas a seguir:

- a) Exibir a mensagem "Enter an integer: ", deixando o cursor na mesma linha.
- b) Atribuir o produto das variáveis *b* e *c* à variável *a*.
- c) Usar um comentário para declarar que um programa efetua um exemplo de cálculo de folha de pagamento.

A.9 Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.

- a) Os operadores Java são avaliados da esquerda para a direita.
- b) Todos os nomes de variável a seguir são válidos: `_under_bar_`, `m928134`, `t5`, `j7`, `her_sales$`, `his_saccount_total`, `a`, `b$`, `c`, `z` e `z2`.
- c) Em Java, uma expressão aritmética válida, sem parênteses, é avaliada da esquerda para a direita.
- d) Todos os nomes de variável a seguir são inválidos: `3g`, `87`, `67h2`, `h22` e `2h`.

A.10 Supondo que $x = 2$ e $y = 3$, o que cada uma das instruções a seguir exibe?

- a) `System.out.printf( "x = %d\n", x );`
- b) `System.out.printf( "Value of %d + %d is %d\n", x, x, ( x + x ) );`
- c) `System.out.printf( "x =" );`
- d) `System.out.printf( "%d = %d\n", ( x + y ), ( y + x ) );`

A.11 (*Aritmética, menor e maior*) Escreva um aplicativo que receba como entrada três valores inteiros do usuário e exiba a soma, a média, o produto, o menor e o maior dos números. Use as técnicas mostradas na Fig. A.15. [Obs.: o cálculo da média neste exercício deve resultar em uma representação inteira da média. Assim, se a soma dos valores for 7, a média deve ser 2, não 2.3333...]

A.12 O que o código a seguir imprime?

```
System.out.printf( "%s\n%s\n%s\n", "☆", "☆☆", "☆☆☆☆" );
```

A.13 (*Maior e menor inteiro*) Escreva um aplicativo que leia cinco valores inteiros e que determine e imprima o maior e o menor do grupo. Use apenas as técnicas de programação que você aprendeu neste apêndice.

A.14 (*Par ou ímpar*) Escreva um aplicativo que leia um valor inteiro e que determine e imprima se ele é par ou ímpar. [Dica: use o operador resto. Um número par é múltiplo de 2. Qualquer múltiplo de 2 deixa um resto 0 quando dividido por 2.]

A.15 (*Múltiplos*) Escreva um aplicativo que leia dois valores inteiros, determine se o primeiro é um múltiplo do segundo e imprima o resultado. [Dica: use o operador resto.]

A.16 (*Diâmetro, circunferência e área de um círculo*) Aqui está um vislumbre do que vem a seguir. Neste apêndice, você aprendeu sobre valores inteiros e o tipo `int`. A linguagem Java também pode representar números de ponto flutuante que contêm casas decimais, como 3,14159. Escreva um aplicativo que receba como entrada do usuário o raio de um círculo como um valor inteiro e imprima o diâmetro, a circunferência e a área do círculo, usando o valor de ponto flutuante 3,14159 para π . Use as técnicas mostradas na Fig. A.7. [Obs.: você também pode usar a constante predefinida `Math.PI` para o valor de π . Essa constante é mais precisa que o valor 3,14159. A classe `Math` está definida no pacote `java.lang`. As classes desse

pacote são importadas automaticamente, de modo que não é necessário importar a classe `Math` para usá-la.] Use as seguintes fórmulas (r é o raio):

$$\begin{aligned}\text{diâmetro} &= 2r \\ \text{circunferência} &= 2\pi r \\ \text{área} &= \pi r^2\end{aligned}$$

Não armazene os resultados de cada cálculo em uma variável. Em vez disso, especifique cada cálculo como o valor que vai ser gerado na saída em uma instrução `System.out.printf`. Os valores produzidos pelos cálculos de circunferência e área são números de ponto flutuante. Tais valores podem ser gerados na saída com o especificador de formato `%f` em uma instrução `System.out.printf`. Você vai aprender mais sobre números de ponto flutuante no Apêndice B.

- A.17 (Separando os algarismos de um número inteiro)** Escreva um aplicativo que receba como entrada do usuário um número composto de cinco algarismos, separe o número em seus dígitos individuais e os imprima separados uns dos outros por três espaços cada um. Por exemplo, se o usuário digitar o número 42339, o programa deverá imprimir

```
4  2  3  3  9
```

Presuma que o usuário digite o número correto de algarismos. O que acontece quando você executa o programa e digita um número com mais de cinco algarismos? O que acontece quando você executa o programa e digita um número com menos de cinco algarismos? [Dica: é possível fazer este exercício com as técnicas que você aprendeu neste apêndice. Você vai aprender a usar operações de divisão e resto para “pinçar” cada algarismo.]

- A.18 (Tabela quadrados e cubos)** Usando somente as técnicas de programação aprendidas neste apêndice, escreva um aplicativo que calcule os quadrados e os cubos dos números de 0 a 10 e imprima os valores resultantes em forma de tabela, como mostrado a seguir. [Obs.: este programa não exige entrada do usuário.]

número	quadrado	cubo
0	0	0
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

Introdução a classes, objetos, métodos e strings

B

Objetivos

Neste capítulo, você vai:

- Aprender a declarar uma classe e a usá-la para criar um objeto.
- Aprender a implementar os comportamentos de uma classe como métodos.
- Aprender a implementar os atributos de uma classe como variáveis de instância e propriedades.
- Aprender a chamar os métodos de um objeto para fazê-los executar suas tarefas.
- Aprender o que são variáveis de instância de uma classe e variáveis locais de um método.
- Aprender a usar um construtor para inicializar os dados de um objeto.
- Conhecer as diferenças entre os tipos primitivos e os tipos de referência.



- | | |
|---|--|
| B.1 Introdução | B.5 Tipos primitivos <i>versus</i> tipos de referência |
| B.2 Declaração de uma classe com um método e instanciação de um objeto de uma classe | B.6 Inicialização de objetos com construtores |
| B.3 Declaração de um método com um parâmetro | B.7 Números de ponto flutuante e o tipo <code>double</code> |
| B.4 Variáveis de instância, métodos <code>set</code> e métodos <code>get</code> | B.8 Para finalizar |

Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios

B.1 Introdução

Neste apêndice, apresentamos alguns conceitos importantes sobre programação orientada a objetos com Java, incluindo classes, objetos, métodos, variáveis de instância e construtores. Exploramos as diferenças entre os tipos primitivos e os tipos de referência, e apresentamos um framework simples para organizar aplicativos voltados para objetos.

B.2 Declaração de uma classe com um método e instanciação de um objeto de uma classe

Nesta seção, você vai criar uma *nova* classe e, então, vai usá-la para criar um objeto. Começamos declarando as classes `GradeBook` (Fig. B.1) e `GradeBookTest` (Fig. B.2). A classe `GradeBook` (declarada no arquivo `GradeBook.java`) vai ser usada para exibir uma mensagem na tela (Fig. B.2) dando as boas-vindas ao professor em um aplicativo de folha de notas (*grade book*). A classe `GradeBookTest` (declarada no arquivo `GradeBookTest.java`) é uma classe de aplicativo na qual o método `main` vai criar e usar um objeto da classe `GradeBook`. *Cada declaração de classe que começa com a palavra-chave `public` deve ser armazenada em um arquivo que tenha o mesmo nome da classe e termine com a extensão .java.* Assim, as classes `GradeBook` e `GradeBookTest` devem ser declaradas em arquivos *separados*, pois cada classe é declarada como `public`.

Classe `GradeBook`

A declaração da classe `GradeBook` (Fig. B.1) contém um método `displayMessage` (linhas 7 a 10) que exibe uma mensagem na tela. Precisaremos produzir um objeto dessa classe e chamar seu método para executar a linha 9 e exibir a mensagem.

```

1 // Fig. B.1: GradeBook.java
2 // Declaração de classe com apenas um método.
3
4 public class GradeBook
5 {
6     // exibe uma mensagem de boas-vindas ao usuário do aplicativo de notas
7     public void displayMessage()
8     {
9         System.out.println( "Welcome to the Grade Book!" );
10    } // fim do método displayMessage
11 } // fim da classe GradeBook

```

Figura B.1 Declaração de classe com apenas um método.

A *declaração da classe* começa na linha 4. A palavra-chave `public` é um **modificador de acesso**. Por enquanto, vamos simplesmente declarar cada classe como `public`. Toda declaração de classe contém a palavra-chave `class`, seguida imediatamente pelo nome da classe. O corpo de toda classe é incluído em um par de chaves de abertura e fechamento, como nas linhas 5 e 11 da classe `GradeBook`.

No Apêndice A, cada classe que declaramos tinha apenas um método, chamado `main`. A classe `GradeBook` também tem apenas um método – `displayMessage` (linhas 7 a 10). Lembre-se de que `main` é um método especial que *sempre* é chamado automaticamente pela JVM (Java Virtual Machine) quando você executa um aplicativo. Em sua maioria, os métodos não são chamados automaticamente. Conforme você verá em breve, deverá chamar o método `displayMessage` explicitamente para fazê-lo executar sua tarefa.

A declaração de método começa com a palavra-chave `public` para indicar que o método está “disponível para o público” – ele pode ser chamado a partir de métodos de outras classes. Em seguida, aparece o **tipo de retorno** do método, o qual especifica o tipo de dado retornado por ele para seu chamador, após executar sua tarefa. O tipo de retorno `void` indica que esse método vai executar uma tarefa, mas *não* vai retornar (isto é, devolver) informação para seu **método chamador**. Você já usou métodos que retornam informações – por exemplo, no Apêndice A, você usou o método `Scanner.nextInt` para inserir na entrada um valor inteiro digitado no teclado pelo usuário. Quando `nextInt` lê um valor do usuário, retorna esse valor para ser usado no programa.

O nome do método, `displayMessage`, segue o tipo de retorno. Por convenção, os nomes de método começam com uma letra minúscula, e as palavras subsequentes do nome começam com letra maiúscula. Os *parênteses* após o nome do método indicam que se trata de um *método*. Parênteses vazios, como na linha 7, indicam que esse método não exige informações adicionais para executar sua tarefa. A linha 7 é comumente referida como **cabeçalho do método**. O corpo de todo método é delimitado por chaves de abertura e fechamento, como nas linhas 8 e 10.

O corpo de um método contém uma ou mais instruções que executam a tarefa do método. Neste caso, o método contém apenas uma instrução (linha 9) que exibe a mensagem “Welcome to the Grade Book!”, seguida de uma nova linha (por causa de `println`) na janela de comando. Após a execução dessa instrução, o método concluiu sua tarefa.

Classe `GradeBookTest`

Em seguida, vamos usar a classe `GradeBook` em um aplicativo. Conforme você aprendeu no Apêndice A, o método `main` inicia a execução de *todo* aplicativo. Uma classe que contém o método `main` inicia a execução de um aplicativo Java. A classe `GradeBook` *não* é um aplicativo, porque *não* contém `main`. Portanto, se você tentar executar `GradeBook` digitando `java GradeBook` na janela de comando, ocorrerá um erro. Para corrigir esse problema, devemos declarar uma classe separada que contenha um método `main` ou colocar um método `main` na classe `GradeBook`. Para ajudá-lo a se preparar para os programas maiores que você vai encontrar mais adiante neste livro e em sua vida profissional, utilizamos uma classe separada (`GradeBookTest` neste exemplo) que contém o método `main`, para testar cada nova classe que criarmos. Alguns programadores se referem a tal classe como *classe driver*. A declaração da classe `GradeBookTest` (Fig. B.2) contém o método `main` que vai controlar a execução de nosso aplicativo.

As linhas 7 a 14 declaram o método `main`. Uma parte importante de habilitar a JVM para localizar e chamar o método `main` para iniciar a execução do aplicativo é a palavra-chave `static` (linha 7), a qual indica que `main` é um método estático. *Um método estático é especial, pois você pode chamá-lo sem primeiro criar um objeto da classe na qual ele é declarado*. Discutimos os métodos estáticos no Apêndice D.

```

1 // Fig. B.2: GradeBookTest.java
2 // Criando um objeto GradeBook e chamando seu método displayMessage.
3
4 public class GradeBookTest
5 {
6     // o método main inicia a execução do programa
7     public static void main( String[] args )
8     {
9         // cria um objeto GradeBook e o atribui a myGradeBook
10        GradeBook myGradeBook = new GradeBook();
11
12        // chama o método displayMessage de myGradeBook
13        myGradeBook.displayMessage();
14    } // fim de main
15 } // fim da classe GradeBookTest

```

Welcome to the Grade Book!

Figura B.2 Criando um objeto GradeBook e chamando seu método displayMessage.

Neste aplicativo, queremos chamar o método `displayMessage` da classe `GradeBook` para exibir a mensagem de boas-vindas na janela de comando. Normalmente não é possível chamar um método que pertence a outra classe até que você crie um objeto dessa classe, como mostrado na linha 10. Começamos declarando a variável `myGradeBook`. O tipo da variável é `GradeBook` — a classe que declaramos na Fig. B.1. Cada nova *classe* que você cria se torna um novo *tipo* que pode ser usado para declarar variáveis e criar objetos.

A variável `myGradeBook` é inicializada (linha 10) com o resultado da **expressão de criação de instância de classe** `new GradeBook()`. A palavra-chave **new** cria um novo objeto da classe especificada no lado direito da palavra-chave (isto é, `GradeBook`). Os parênteses no lado direito de `GradeBook` são obrigatórios. Conforme você vai aprender na seção B.6, esses parênteses, combinados com um nome de classe, representam uma chamada para um **construtor**, o qual é semelhante a um método, mas usado no momento em que um objeto é *criado* para *inicializar* os dados do objeto. Você vai ver que informações podem ser colocadas nos parênteses para especificar *valores iniciais* para os dados do objeto. Por enquanto, vamos simplesmente deixar os parênteses vazios.

Assim como podemos usar o objeto `System.out` para chamar seus métodos `print`, `printf` e `println`, podemos usar o objeto `myGradeBook` para chamar seu método `displayMessage`. A linha 13 chama o método `displayMessage` (linhas 7 a 10 da Fig. B.1) usando `myGradeBook`, seguido por um **separador ponto** (`.`), o nome do método `displayMessage` e um conjunto de parênteses vazios. Essa chamada faz o método `displayMessage` executar sua tarefa. Essa chamada de método é diferente daquelas do Apêndice A, que exibiam informações em uma janela de comando — cada uma daquelas chamadas de método fornecia argumentos que especificavam os dados a serem exibidos. No início da linha 13, “`myGradeBook.`” indica que `main` deve usar o objeto `myGradeBook` criado na linha 10. A linha 7 da Fig. B.1 indica que o método `displayMessage` tem uma *lista de parâmetros vazia* — isto é, `displayMessage` não exige informações adicionais para executar sua tarefa. Por isso, a chamada de método (linha 13 da Fig. B.2) especifica um conjunto de parênteses vazio após o nome do método para indicar que *nenhum argumento* está sendo passado para o método `displayMessage`. Quando o método `displayMessage` conclui sua tarefa, o método `main` continua a execução na linha 14. Esse é o fim do método `main`, de modo que o programa termina.

Qualquer classe pode conter um método `main`. A JVM chama o método `main` *somente* na classe usada para executar o aplicativo. Se um aplicativo tem várias classes que contêm `main`, o método chamado é aquele que está na classe nomeada no comando `java`.

Compilando um aplicativo com várias classes

É preciso compilar as classes da Fig. B.1 e da Fig. B.2 antes que você possa executar o aplicativo. Primeiramente, mude para o diretório que contém os arquivos de código-fonte do aplicativo. Em seguida, digite o comando

```
javac GradeBook.java GradeBookTest.java
```

para compilar *as duas* classes simultaneamente. Se o diretório que contém o aplicativo inclui somente os arquivos desse aplicativo, você pode compilar *todas* as classes do diretório com o comando

```
javac *.java
```

O asterisco (*) em `*.java` indica que devem ser compilados *todos* os arquivos do diretório atual que terminam com a extensão `".java"`.

B.3 Declaração de um método com um parâmetro

Em nossa analogia do carro, da seção 1.8, discutimos o fato de que pressionar o acelerador envia uma *mensagem* para o carro executar uma *tarefa* – ou seja, ir mais rápido. Mas *quanto* se deve acelerar? Como você sabe, quanto mais se pressiona o acelerador, mais o carro acelera. Assim, a mensagem para o carro inclui a *tarefa a ser executada* e a *informação adicional* que ajuda o carro a executá-la. Essa informação adicional é conhecida como **parâmetro** – o valor do parâmetro ajuda o carro a determinar quanto se deve acelerar. Analogamente, um método exibe um ou mais parâmetros que representam as informações adicionais de que ele necessita para executar sua tarefa. Os parâmetros são definidos em uma **lista de parâmetros** separados por vírgulas, a qual fica dentro dos parênteses que seguem o nome do método. Cada parâmetro deve especificar um *tipo* e um nome de variável. A lista de parâmetros pode conter qualquer número de parâmetros, inclusive nenhum. Parênteses vazios após o nome do método (como na Fig. B.1, linha 7) indicam que o método *não* exige parâmetros.

Argumentos para um método

Uma chamada de método fornece valores – denominados *argumentos* – para cada um dos parâmetros do método. Por exemplo, o método `System.out.println` exige um argumento que especifica os dados a serem gerados na saída em uma janela de comando. Do mesmo modo, para fazer um depósito em uma conta bancária, um método `deposit` especifica um parâmetro que representa o valor do depósito. Quando o método `deposit` é chamado, um valor de argumento representando o valor do depósito é atribuído ao parâmetro do método. Então, o método faz um depósito nesse valor.

Declaração de classe com um método que tem um parâmetro

Agora, declaramos a classe `GradeBook` (Fig. B.3) com um método `displayMessage` que exibe o nome do curso como parte da mensagem de boas-vindas. (Veja o exemplo de execução na Fig. B.4.) O novo método exige um parâmetro que represente o nome do curso na saída.

Antes de discutirmos os novos recursos da classe `GradeBook`, vamos ver como a nova classe é usada a partir do método `main` da classe `GradeBookTest` (Fig. B.4). A linha 12 cria um objeto `Scanner` chamado `input` para ler o nome do curso do usuário. A linha 15 cria o objeto `myGradeBook` de `GradeBook`. A linha 18 pede ao usuário para que digite um nome de curso. A linha 19 lê o nome digitado pelo usuário e o atribui à variável `nameOfCourse`, usando o método `nextLine` de `Scanner` para fazer a entrada. O usuário digita o nome do curso e pressiona *Enter* para enviá-lo ao programa. Pressionar *Enter* insere um caractere de nova linha no final dos caracteres digitados pelo usuário. O método `nextLine` lê os caracteres digitados pelo usuário até encontrar o caractere de nova linha; então, retorna um objeto `String` contendo os caracteres até o de nova linha, mas *não* o incluindo. O caractere de nova linha é *descartado*.

```

1 // Fig. B.3: GradeBook.java
2 // Declaração de classe com um método que tem um parâmetro.
3
4 public class GradeBook
5 {
6     // exibe uma mensagem de boas-vindas ao usuário do aplicativo de notas
7     public void displayMessage( String courseName )
8     {
9         System.out.printf( "Welcome to the grade book for\n%s!\n",
10             courseName );
11     } // fim do método displayMessage
12 } // fim da classe GradeBook

```

Figura B.3 Declaração de classe com um método que tem um parâmetro.

```

1 // Fig. B.4: GradeBookTest.java
2 // Cria um objeto GradeBook e passa uma String para
3 // seu método displayMessage.
4 import java.util.Scanner; // o programa usa Scanner
5
6 public class GradeBookTest
7 {
8     // o método main inicia a execução do programa
9     public static void main( String[] args )
10    {
11        // cria um objeto Scanner para obter entrada da janela de comando
12        Scanner input = new Scanner( System.in );
13
14        // cria um objeto GradeBook e o atribui a myGradeBook
15        GradeBook myGradeBook = new GradeBook();
16
17        // solicita e insere o nome do curso
18        System.out.println( "Please enter the course name:" );
19        String nameOfCourse = input.nextLine(); // lê uma linha de texto
20        System.out.println(); // gera uma linha em branco na saída
21
22        // chama o método displayMessage de myGradeBook
23        // e passa nameOfCourse como argumento
24        myGradeBook.displayMessage( nameOfCourse );
25    } // fim de main
26 } // fim da classe GradeBookTest

```

Figura B.4 Cria um objeto `GradeBook` e passa uma `String` para seu método `displayMessage`.
(continua)

```
Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
```

Figura B.4 Cria um objeto GradeBook e passa uma String para seu método `displayMessage`.

A classe `Scanner` fornece também o método `next`, o qual lê palavras individuais. Quando o usuário pressiona *Enter* depois de digitar a entrada, o método `next` lê os caracteres até encontrar um *caractere de espaço em branco* (como um espaço, tabulação ou nova linha) e, então, retorna uma `String` contendo os caracteres até o de espaço em branco (que é descartado), mas *não o* incluindo. Toda informação após o primeiro caractere de espaço em branco não é perdida – ela pode ser lida por outras instruções que chamem os métodos de `Scanner` mais adiante no programa. A linha 20 é uma linha em branco.

A linha 24 chama o método `displayMessage` de `myGradeBooks`. A variável `nameOfCourse` nos parênteses é o *argumento* passado para o método `displayMessage` para que ele possa executar sua tarefa. O valor da variável `nameOfCourse` em `main` se torna o valor do *parâmetro* `courseName` do método `displayMessage` na linha 7 da Fig. B.3. Quando esse aplicativo é executado, observe que o método `displayMessage` gera na saída o nome que você digita como parte da mensagem de boas-vindas (Fig. B.4).

Detalhes adicionais sobre argumentos e parâmetros

Na Fig. B.3, a lista de parâmetros de `displayMessage` (linha 7) declara um parâmetro indicando que o método exige uma `String` para executar sua tarefa. Quando o método é chamado, um valor de argumento na chamada é atribuído ao parâmetro correspondente (`courseName`) no cabeçalho do método. Então, o corpo do método usa o valor do parâmetro `courseName`. As linhas 9 e 10 da Fig. B.3 exibem o valor do parâmetro `courseName`, usando o especificador de formato `%s` na string de formato de `printf`. O nome da variável do parâmetro (`courseName` na Fig. B.3, linha 7) pode ser *o mesmo ou diferente* do nome da variável do argumento (`nameOfCourse` na Fig. B.4, linha 24).

O número de argumentos em uma chamada de método *precisa* corresponder ao número de parâmetros da lista de parâmetros da declaração do método. Além disso, os tipos dos argumentos na chamada de método devem ser “coerentes com” os tipos dos parâmetros correspondentes na declaração do método. (Conforme você vai aprender no Apêndice D, o tipo de um argumento e o tipo de seu parâmetro correspondente nem sempre precisam ser *idênticos*.) Em nosso exemplo, a chamada de método passa um argumento de tipo `String` (`nameOfCourse` é declarado como `String` na linha 19 da Fig. B.4) e a declaração do método especifica um parâmetro de tipo `String` (`courseName` é declarado como `String` na linha 7 da Fig. B.3). Assim, nesse exemplo, o tipo do argumento na chamada de método corresponde exatamente ao tipo do parâmetro no cabeçalho do método.

Observações sobre as declarações `import`

Observe a declaração `import` na Fig. B.4 (linha 4). Isso indica ao compilador que o programa usa a classe `Scanner`. Por que precisamos importar a classe `Scanner`, mas não as classes `System`, `String` ou `GradeBook`? As classes `System` e `String` estão no pacote `java.lang`, o qual é importado implicitamente para *todo* programa Java; portanto, todos os programas

podem usar as classes desse pacote *sem* importá-las explicitamente. A maioria das outras classes que você vai usar em programas Java deve ser importada explicitamente.

Existe uma relação especial entre as classes compiladas no mesmo diretório no disco, como as classes `GradeBook` e `GradeBookTest`. Por padrão, essas classes são consideradas como estando no mesmo pacote – conhecido como **pacote padrão**. As classes do mesmo pacote são *importadas implicitamente* para os arquivos de código-fonte de outras classes no mesmo pacote. Assim, *não* é exigida uma declaração `import` quando uma classe de um pacote utiliza outra do mesmo pacote – como quando a classe `GradeBookTest` utiliza a classe `GradeBook`.

A declaração `import` na linha 4 *não* é exigida, caso sempre façamos referência à classe `Scanner` como `java.util.Scanner`, que inclui o *nome do pacote completo e o nome da classe*. Isso é conhecido como **nome de classe totalmente qualificado**. Por exemplo, a linha 12 poderia ser escrita como

```
java.util.Scanner input = new java.util.Scanner( System.in );
```

B.4 Variáveis de instância, métodos `set` e métodos `get`

No Apêndice A, declaramos todas as variáveis de um aplicativo no método `main`. As variáveis declaradas no corpo de um método em particular são conhecidas como **variáveis locais** e só podem ser usadas nesse método. Quando esse método termina, os valores de suas variáveis locais são perdidos. Lembre que na Seção 1.8 um objeto tem *atributos* que carrega consigo ao ser usado em um programa. Esses atributos existem antes que um método seja chamado em um objeto, enquanto o método está sendo executado e depois que termina de ser executado.

Normalmente, uma classe consiste em um ou mais métodos que manipulam os atributos pertencentes a um objeto em particular da classe. Em uma declaração de classe, os atributos são representados como variáveis. Essas variáveis são denominadas **campos** e são declaradas *dentro* de uma declaração de classe, mas *fora* dos corpos das declarações de método da classe. Quando cada objeto de uma classe mantém sua própria cópia de um atributo, o campo que representa o atributo também é conhecido como **variável de instância** – cada objeto (instância) da classe tem uma instância separada da variável na memória. O exemplo desta seção demonstra uma classe `GradeBook` que contém uma variável de instância `courseName` para representar o nome do curso de um objeto `GradeBook` específico.

Classe `GradeBook` com uma variável de instância, um método `set` e um método `get`

Em nosso próximo aplicativo (Figs. B.5 e B.6), a classe `GradeBook` (Fig. B.5) mantém o nome do curso como uma variável de instância para que possa ser usado ou modificado a qualquer momento durante a execução do aplicativo. A classe contém três métodos – `setCourseName`, `getCourseName` e `displayMessage`. O método `setCourseName` armazena um nome de curso em um objeto `GradeBook`. O método `getCourseName` obtém um nome de curso de `GradeBook`. O método `displayMessage`, que agora não especifica um parâmetro, ainda exibe uma mensagem de boas-vindas que inclui o nome do curso; como você vai ver, agora o método obtém o nome do curso chamando um método na mesma classe – `getCourseName`.

```

1 // Fig. B.5: GradeBook.java
2 // Classe GradeBook que contém uma variável de instância courseName
3 // e métodos para configurar e obter seu valor.
4
5 public class GradeBook
6 {
7     private String courseName; // nome do curso para este objeto GradeBook
8
9     // método para definir o nome do curso
10    public void setCourseName( String name )
11    {
12        courseName = name; // armazena o nome do curso
13    } // fim do método setCourseName
14
15    // método para recuperar o nome do curso
16    public String getCourseName()
17    {
18        return courseName;
19    } // fim do método getCourseName
20
21    // exibe uma mensagem de boas-vindas ao usuário do aplicativo de notas
22    public void displayMessage()
23    {
24        // chama getCourseName para obter o nome do
25        // curso representado por esse objeto GradeBook
26        System.out.printf( "Welcome to the grade book for\n%s!\n",
27                           getCourseName() );
28    } // fim do método displayMessage
29 } // fim da classe GradeBook

```

Figura B.5 Classe GradeBook que contém uma variável de instância courseName e métodos para configurar e obter seu valor.

Normalmente, um professor dá aulas em mais de um curso, e cada curso tem seu próprio nome. A linha 7 declara courseName como uma variável de tipo String. Como a variável é declarada *no* corpo da classe, mas *fora* dos corpos dos métodos da classe (linhas 10 a 13, 16 a 19 e 22 a 28), a linha 7 é a declaração de uma *variável de instância*. Toda instância (isto é, objeto) da classe GradeBook contém uma cópia de cada variável de instância. Por exemplo, se houver dois objetos GradeBook, cada objeto terá sua própria cópia de courseName. Uma vantagem de tornar courseName uma variável de instância é que todos os métodos da classe (neste caso, GradeBook) podem manipular quaisquer variáveis de instância que apareçam na classe (neste caso, courseName).

Modificadores de acesso public e private

A maioria das declarações de variável de instância é precedida pela palavra-chave private (como na linha 7). Assim como public, a palavra-chave **private** é um *modificador de acesso*. Variáveis ou métodos declarados com o modificador de acesso private são *acessíveis somente nos métodos da classe em que são declarados*. Assim, a variável courseName só pode ser usada nos métodos setCourseName, getCourseName e displayMessage da (de cada objeto da) classe GradeBook.

Declarar variáveis de instância com o modificador de acesso private é conhecido como **ocultação de dados** ou ocultação de informação. Quando um programa cria (instancia) um objeto da classe GradeBook, a variável courseName é *encapsulada* (oculta) no objeto e só pode ser acessada por métodos da classe do objeto. Isso impede que courseName seja modificada acidentalmente por uma classe em outra parte do programa. Na classe GradeBook, os métodos setCourseName e getCourseName manipulam a variável de instância courseName.



Observação sobre engenharia de software B.1

Preceda cada campo e declaração do método com um modificador de acesso. Geralmente, as variáveis de instância devem ser declaradas como `private` e os métodos como `public`. (É apropriado declarar certos métodos como `private`, caso devam ser acessados somente por outros métodos da classe.)

Métodos `setCourseName` e `getCourseName`

O método `setCourseName` (linhas 10 a 13) não retorna nenhum dado ao completar sua tarefa, de modo que seu tipo de retorno é `void`. O método recebe um único parâmetro – `name` – representando o nome do curso que vai ser passado a ele como argumento. A linha 12 atribui `name` à variável de instância `courseName`.

O método `getCourseName` (linhas 16 a 19) retorna a variável `courseName` de um objeto `GradeBook` em particular. O método tem uma lista de parâmetros vazia, de modo que não exige informações adicionais para executar sua tarefa. Ele especifica que retorna uma `String` – esse é o tipo de retorno do método. Quando um método que especifica um tipo de retorno que não é `void` é chamado e conclui sua tarefa, retorna um *resultado* para seu método chamador. Por exemplo, quando você se dirige a um caixa eletrônico (ATM) e solicita o saldo de sua conta, espera receber um valor que represente seu saldo. Do mesmo modo, quando uma instrução chama o método `getCourseName` em um objeto `GradeBook`, ela espera receber o nome do curso de `GradeBook` (neste caso, uma `String`, conforme especificado no tipo de retorno da declaração do método).

A instrução **`return`** na linha 18 passa o valor da variável de instância `courseName` de volta para a instrução que chama o método `getCourseName`. Considere a linha 27 do método `displayMessage`, a qual chama o método `getCourseName`. Quando o valor é retornado, a instrução nas linhas 26 e 27 utiliza esse valor para gerar o nome do curso na saída. Da mesma forma, se você tivesse um método `square` que retornasse o quadrado de seu argumento, esperaria que a instrução

```
int result = square( 2 );
```

retornasse 4 do método `square` e atribuísse 4 à variável `result`. Se tivesse um método `maximum` que retornasse o maior de três argumentos inteiros, esperaria que a instrução

```
int biggest = maximum( 27, 114, 51 );
```

retornasse 114 do método `maximum` e atribuísse 114 à variável `biggest`.

As instruções nas linhas 12 e 18 utilizam `courseName` *mesmo não sendo declarada em nenhum desses métodos*. Podemos usar `courseName` nos métodos de `GradeBook` porque `courseName` é uma variável de instância da classe.

Método `displayMessage`

O método `displayMessage` (linhas 22 a 28) *não* retorna nenhum dado ao completar sua tarefa, de modo que seu tipo de retorno é `void`. O método *não* recebe parâmetros, de modo que a lista de parâmetros está vazia. As linhas 26 e 27 geram na saída uma mensagem de boas-vindas que inclui o valor da variável de instância `courseName`, o qual é retornado pela chamada ao método `getCourseName` na linha 27. Observe que um método de uma classe (`displayMessage` neste caso) pode chamar outro método da *mesma* classe usando apenas o nome do método (`getCourseName` neste caso).

Classe `GradeBookTest` que demonstra a classe `GradeBook`

A classe `GradeBookTest` (Fig. B.6) cria um objeto da classe `GradeBook` e demonstra seus métodos. A linha 14 cria um objeto `GradeBook` e o atribui à variável local `myGradeBook` de

```

1 // Fig. B.6: GradeBookTest.java
2 // Criando e manipulando um objeto GradeBook.
3 import java.util.Scanner; // o programa usa Scanner
4
5 public class GradeBookTest
6 {
7     // o método main inicia a execução do programa
8     public static void main( String[] args )
9     {
10         // cria um objeto Scanner para obter entrada da janela de comando
11         Scanner input = new Scanner( System.in );
12
13         // cria um objeto GradeBook e o atribui a myGradeBook
14         GradeBook myGradeBook = new GradeBook();
15
16         // exibe o valor inicial de courseName
17         System.out.printf( "Initial course name is: %s\n\n",
18             myGradeBook.getCourseName() );
19
20         // solicita e lê o nome do curso
21         System.out.println( "Please enter the course name:" );
22         String theName = input.nextLine(); // lê uma linha de texto
23         myGradeBook.setCourseName( theName ); // configura o nome do curso
24         System.out.println(); // gera uma linha em branco na saída
25
26         // exibe mensagem de boas-vindas após especificar o nome do curso
27         myGradeBook.displayMessage();
28     } // fim de main
29 } // fim da classe GradeBookTest

```

```

Initial course name is: null

Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!

```

Figura B.6 Criando e manipulando um objeto GradeBook.

tipo GradeBook. As linhas 17 e 18 exibem o nome de curso inicial, chamando o método `getCourseName` do objeto. A primeira linha da saída mostra o nome “null”. *Ao contrário das variáveis locais, que não são inicializadas automaticamente, todo campo tem um **valor inicial padrão** – um valor fornecido pela linguagem Java quando você não especifica o valor inicial do campo.* Assim, não é obrigatório inicializar os campos explicitamente antes de serem usados em um programa – a não ser que devam ser inicializados com valores diferentes de seus valores padrão. O valor padrão para um campo de tipo String (como `courseName` neste exemplo) é `null`, sobre o qual damos mais informações na Seção B.5.

A linha 21 pede ao usuário para que digite um nome de curso. A variável String local `theName` (declarada na linha 22) é inicializada com o nome de curso digitado pelo usuário, o qual é retornado pela chamada ao método `nextLine` do objeto Scanner `input`. A linha 23 chama o método `setCourseName` do objeto `myGradeBook` e fornece `theName` como argumento do método. Quando o método é chamado, o valor do argumento é atribuído ao parâmetro `name` (linha 10, Fig. B.5) do método `setCourseName` (linhas 10 a 13, Fig. B.5). Então, o valor do parâmetro é atribuído à variável de instância `courseName` (linha 12, Fig. B.5). A linha 24 (Fig. B.6) pula uma linha na saída e a linha 27 chama o método `displayMessage` do objeto `myGradeBook` para exibir a mensagem de boas-vindas contendo o nome do curso.

Métodos set e get

Os campos `private` de uma classe só podem ser manipulados pelos métodos da classe. Portanto, o **cliente de um objeto** – isto é, qualquer classe que chame os métodos do objeto – chama os métodos `public` da classe para manipular os campos `private` de um objeto da classe. É por isso que as instruções no método `main` (Fig. B.6) chamam os métodos `setCourseName`, `getCourseName` e `displayMessage` em um objeto `GradeBook`. As classes frequentemente fornecem métodos `public` para permitir aos clientes configurar – **set** – (isto é, atribuir valores a) ou obter – **get** – (isto é, receber os valores de) variáveis de instância `private`. Os nomes desses métodos não precisam começar com *set* ou *get*, mas essa convenção de atribuição de nomes é recomendada e é uma convenção para componentes de software Java especiais, chamados JavaBeans, os quais podem simplificar a programação em muitos ambientes de desenvolvimento integrados (IDEs) Java. O método que *configura* a variável de instância `courseName` neste exemplo é chamado `setCourseName`, e o método que *obtem* seu valor é chamado `getCourseName`.

B.5 Tipos primitivos versus tipos de referência

A linguagem Java tem tipos primitivos e **tipos de referência**. Os tipos primitivos são: `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` e `double`. Todos os que não são primitivos são tipos de referência; portanto, as classes, que especificam os tipos dos objetos, são tipos de referência.

Uma variável de tipo primitivo pode armazenar exatamente um *valor de seu tipo declarado* por vez. Por exemplo, uma variável `int` pode armazenar um número inteiro (como 7) por vez. Quando outro valor é atribuído a essa variável, seu valor inicial é substituído. As variáveis de instância de tipo primitivo são *inicializadas por padrão* – variáveis dos tipos `byte`, `char`, `short`, `int`, `long`, `float` e `double` são inicializadas com 0 e variáveis de tipo `boolean` são inicializadas com `false`. Você pode especificar seu próprio valor inicial para uma variável de tipo primitivo, atribuindo a ela um valor em sua declaração, como em

```
private int numberOfStudents = 10;
```

Lembre-se de que as variáveis locais *não* são inicializadas por padrão.



Dica de prevenção de erro B.1

Usar uma variável local não inicializada causa um erro de compilação.

Os programas utilizam variáveis de tipos de referência (normalmente chamadas de **referências**) para armazenar o *local* dos objetos na memória do computador. Diz-se que tais variáveis **fazem referência a um objeto** no programa. Cada um dos objetos referenciados pode conter muitas variáveis de instância. A linha 14 da Fig. B.6 cria um objeto da classe `GradeBook`, e a variável `myGradeBook` contém uma referência para esse objeto `GradeBook`. *As variáveis de tipo de referência são inicializadas com o valor `null` por padrão* – uma palavra reservada que representa uma “referência para nada”. É por isso que a primeira chamada para `getCourseName` na linha 18 da Fig. B.6 retornou `null` – o valor de `courseName` não havia sido definido, de modo que foi retornado o valor padrão inicial, `null`.

Quando você usa um objeto de outra classe, é necessária uma referência para o objeto a fim de **invocar** (isto é, chamar) seus métodos. No aplicativo da Fig. B.6, as ins-

truções no método `main` utilizam a variável `myGradeBook` a fim de enviar mensagens para o objeto `GradeBook`. Essas mensagens são chamadas para métodos (como `setCourseName` e `getCourseName`) que permitem ao programa interagir com o objeto `GradeBook`. Por exemplo, a instrução na linha 23 usa `myGradeBook` para enviar a mensagem de `setCourseName` para o objeto `GradeBook`. Essa mensagem inclui o argumento exigido por `setCourseName` para realizar sua tarefa. O objeto `GradeBook` utiliza essa informação para configurar a variável de instância de `courseName`. As variáveis de tipo primitivo não fazem referência a objetos, de modo que não podem ser usadas para chamar métodos.



Observação sobre engenharia de software B.2

O tipo declarado de uma variável (por exemplo, `int`, `double` ou `GradeBook`) indica se ela é de tipo primitivo ou de tipo de referência. Se uma variável não é de um dos oito tipos primitivos, então é de um tipo de referência.

B.6 Inicialização de objetos com construtores

Como mencionado na seção B.4, quando um objeto da classe `GradeBook` (Fig. B.5) é criado, sua variável de instância `courseName` é inicializada com `null` por padrão. E se você quiser fornecer um nome de curso ao criar um objeto `GradeBook`? Cada classe que você declara pode fornecer um método especial, chamado construtor, que pode ser usado para inicializar um objeto de uma classe quando o objeto é criado. Na verdade, a linguagem Java *exige* uma chamada para o construtor de *todo* objeto criado. A palavra-chave `new` solicita memória do sistema para armazenar um objeto e, então, chama o construtor da classe correspondente para inicializar o objeto. A chamada é indicada pelos parênteses após o nome da classe. Um construtor *deve* ter o *mesmo nome* da classe. Por exemplo, a linha 14 da Fig. B.6 primeiramente usa `new` para criar um objeto `GradeBook`. Os parênteses vazios após “`new GradeBook`” indicam uma chamada para o construtor da classe, sem argumentos. O compilador fornece um **construtor padrão** *sem parâmetros* em qualquer classe que *não* inclua um construtor explicitamente. Quando uma classe tem apenas o construtor padrão, suas variáveis de instância são inicializadas com seus *valores padrão*.

Ao declarar uma classe, você pode fornecer seu próprio construtor para especificar uma inicialização personalizada para os objetos de sua classe. Por exemplo, talvez você queira especificar um nome de curso para o objeto `GradeBook` quando o objeto for criado, como em

```
GradeBook myGradeBook =
    new GradeBook( "CS101 Introduction to Java Programming" );
```

Nesse caso, o argumento “`CS101 Introduction to Java Programming`” é passado para o construtor do objeto `GradeBook` e usado para inicializar `courseName`. A instrução anterior exige que a classe forneça um construtor com um parâmetro de tipo `String`. A Figura B.7 contém uma classe `GradeBook` modificada com um construtor assim.

```
1 // Fig. B.7: GradeBook.java
2 // Classe GradeBook com um construtor para inicializar o nome do curso.
3
4 public class GradeBook
5 {
6     private String courseName; // nome do curso para este objeto GradeBook
7 }
```

Figura B.7 Classe `GradeBook` com um construtor para inicializar o nome do curso. (continua)

```

8      // o construtor inicializa courseName com um argumento String
9      public GradeBook( String name ) // o nome do construtor é o nome da classe
10     {
11         courseName = name; // inicializa courseName
12     } // fim do construtor
13
14     // método para definir o nome do curso
15     public void setCourseName( String name )
16     {
17         courseName = name; // armazena o nome do curso
18     } // fim do método setCourseName
19
20     // método para recuperar o nome do curso
21     public String getCourseName()
22     {
23         return courseName;
24     } // fim do método getCourseName
25
26     // exibe uma mensagem de boas-vindas ao usuário do aplicativo de notas
27     public void displayMessage()
28     {
29         // esta instrução chama getCourseName para obter o
30         // nome do curso representado por esse objeto GradeBook
31         System.out.printf( "Welcome to the grade book for\n%s!\n",
32             getCourseName() );
33     } // fim do método displayMessage
34 } // fim da classe GradeBook

```

Figura B.7 Classe GradeBook com um construtor para inicializar o nome do curso.

As linhas 9 a 12 declaram o construtor de GradeBook. Assim como um método, a lista de parâmetros de um construtor especifica os dados exigidos por ele para executar sua tarefa. Quando você cria um novo objeto (como vamos fazer na Fig. B.8), esses dados são colocados nos *parênteses que seguem o nome da classe*. A linha 9 da Fig. B.7 indica que o construtor tem um parâmetro String chamado name. O parâmetro name passado para o construtor é atribuído à variável de instância courseName na linha 11.

A Figura B.8 inicializa objetos de GradeBook usando o construtor. As linhas 11 e 12 criam e inicializam o objeto gradeBook1 de GradeBook. O construtor de GradeBook é chamado com o argumento "CS101 Introduction to Java Programming" para inicializar o nome do curso. A expressão de criação da instância da classe nas linhas 11 e 12 retorna uma referência para o novo objeto, a qual é atribuída à variável gradeBook1. As linhas 13 e 14 repetem esse processo, desta vez passando o argumento "CS102 Data Structures in Java" para inicializar o nome do curso para gradeBook2. As linhas 17 a 20 usam o método getCourseName de cada objeto para obter os nomes de curso e mostrar que eles foram inicializados quando os objetos foram criados. A saída confirma que cada objeto GradeBook mantém sua própria cópia da variável de instância courseName.

Uma diferença importante entre construtores e métodos é que os construtores não podem retornar valores; portanto, não podem especificar um tipo de retorno (nem mesmo void). Normalmente, os construtores são declarados public. Se uma classe não inclui um construtor padrão, suas variáveis de instância são inicializadas com seus valores padrão. *Se você declarar quaisquer construtores para uma classe, o compilador Java não vai criar um construtor padrão para essa classe.* Assim, não podemos mais criar um objeto GradeBook com new GradeBook(), como fizemos nos exemplos anteriores.


```

1 // Fig. B.8: GradeBookTest.java
2 // Construtor de GradeBook usado para especificar o nome do curso no
3 // momento em que cada objeto GradeBook é criado.
4
5 public class GradeBookTest
6 {
7     // o método main inicia a execução do programa
8     public static void main( String[] args )
9     {
10         // cria o objeto GradeBook
11         GradeBook gradeBook1 = new GradeBook(
12             "CS101 Introduction to Java Programming" );
13         GradeBook gradeBook2 = new GradeBook(
14             "CS102 Data Structures in Java" );
15
16         // exibe o valor inicial de courseName para cada GradeBook
17         System.out.printf( "gradeBook1 course name is: %s\n",
18             gradeBook1.getCourseName() );
19         System.out.printf( "gradeBook2 course name is: %s\n",
20             gradeBook2.getCourseName() );
21     } // fim de main
22 } // fim da classe GradeBookTest

```

```

gradeBook1 course name is: CS101 Introduction to Java Programming
gradeBook2 course name is: CS102 Data Structures in Java

```

Figura B.8 Construtor de GradeBook usado para especificar o nome do curso no momento em que cada objeto GradeBook é criado.

Construtores com vários parâmetros

Às vezes, você vai querer inicializar objetos com vários itens de dados. No Exercício B.11, pedimos que você armazene o nome do curso *e* o nome do professor em um objeto GradeBook. Nesse caso, o construtor de GradeBook seria modificado para receber dois objetos String, como em

```
public GradeBook( String courseName, String instructorName )
```

e você chamaria o construtor de GradeBook como segue:

```
GradeBook gradeBook = new GradeBook(
    "CS101 Introduction to Java Programming", "Sue Green" );
```

B.7 Números de ponto flutuante e o tipo double

Agora, vamos deixar de lado o estudo de caso de GradeBook temporariamente para declarar uma classe Account que mantém o saldo de uma conta bancária. A maioria dos saldos em conta não é constituída de números inteiros (como 0, -22 e 1024). Por isso, a classe Account representa o saldo da conta como um **número de ponto flutuante** (isto é, um número com um ponto decimal, como em 7.33, 0.0975 ou 1000.12345). A linguagem Java fornece dois tipos primitivos para armazenar números de ponto flutuante na memória – float e double. A diferença principal entre os dois é que as variáveis double podem armazenar números de valor maior e mais detalhados (isto é, mais dígitos à direita do ponto decimal – também conhecida como a **precisão** do número) do que as variáveis float.

Precisão e requisitos de memória dos números de ponto flutuante

As variáveis de tipo `float` representam **números de ponto flutuante de precisão simples** e podem representar até *sete dígitos significativos*. As variáveis de tipo `double` representam **números de ponto flutuante de precisão dupla**. Elas exigem duas vezes mais memória que as variáveis `float` e fornecem *15 dígitos significativos* – aproximadamente duas vezes a precisão das variáveis `float`. Para o intervalo de valores exigido pela maioria dos programas, as variáveis de tipo `float` devem ser suficientes, mas você pode usar `double` para “não se arriscar”. Em alguns aplicativos, até as variáveis `double` serão inadequadas. A maioria dos programadores representa números de ponto flutuante com o tipo `double`. Na verdade, por padrão, a linguagem Java trata como valores `double` todos os números de ponto flutuante digitados no código-fonte de um programa (como 7.33 e 0.0975). Esses valores no código-fonte são conhecidos como **literais de ponto flutuante**. Consulte o Apêndice L para ver os intervalos de valores `float` e `double`.

Embora os números de ponto flutuante nem sempre sejam 100% precisos, eles têm numerosas aplicações. Por exemplo, quando falamos da temperatura “normal” do corpo, de 37 graus, não precisamos ser precisos com um número de dígitos maior. Quando lemos a temperatura de 37 graus em um termômetro, na verdade ela pode ser de 36,9999999. Indicar esse número simplesmente como 37 está bem para a maioria das aplicações que envolvem temperaturas corporais. Devido à natureza imprecisa dos números de ponto flutuante, prefere-se o tipo `double` em detrimento do tipo `float`, pois as variáveis `double` podem representar números de ponto flutuante com mais precisão. Por isso, utilizamos principalmente o tipo `double` por todo o livro. Para números de ponto flutuante precisos, a linguagem Java fornece a classe `BigDecimal` (pacote `java.math`).

Os números de ponto flutuante também surgem como resultado de divisões. Na aritmética convencional, quando dividimos 10 por 3, o resultado é 3,333333..., com a sequência de números 3 se repetindo infinitamente. O computador aloca somente uma quantidade fixa de espaço para armazenar um valor assim; portanto, claramente o valor de ponto flutuante armazenado só pode ser uma aproximação.

Classe Account com uma variável de instância de tipo double

Nosso próximo aplicativo (Figs. B.9 e B.10) contém uma classe chamada `Account` (Fig. B.9) que mantém o saldo de uma conta bancária. Um banco normalmente atende a muitas contas, cada uma com seu saldo, de modo que a linha 7 declara uma variável de instância chamada `balance` de tipo `double`. Trata-se de uma variável de instância porque é declarada no corpo da classe, mas fora das declarações de método da classe (linhas 10 a 16, 19 a 22 e 25 a 28). Toda instância (isto é, objeto) da classe `Account` contém sua própria cópia de `balance`.

A classe tem um construtor e dois métodos. É comum alguém abrir uma conta para depositar dinheiro imediatamente, de modo que o construtor (linhas 10 a 16) recebe um parâmetro `initialBalance` de tipo `double` que representa o *saldo inicial*. As linhas 14 e 15 garantem que `initialBalance` seja maior que 0.0. Se assim for, o valor de `initialBalance` é atribuído à variável de instância `balance`. Caso contrário, `balance` permanece em 0.0 – seu valor padrão inicial.

```

1 // Fig. B.9: Account.java
2 // Classe Account com um construtor para validar e
3 // inicializar a variável de instância balance de tipo double.
4
5 public class Account
6 {
7     private double balance; // variável de instância que armazena o saldo
8
9     // construtor
10    public Account( double initialBalance )
11    {
12        // valida o fato de que initialBalance é maior que 0.0;
13        // se não for, balance é inicializada com o valor padrão 0.0
14        if ( initialBalance > 0.0 )
15            balance = initialBalance;
16    } // fim do construtor de Account
17
18    // credita (adiciona) um valor à conta
19    public void credit( double amount )
20    {
21        balance = balance + amount; // soma o valor ao saldo
22    } // fim do método credit
23
24    // retorna o saldo da conta
25    public double getBalance()
26    {
27        return balance; // fornece o valor do saldo para o método chamador
28    } // fim do método getBalance
29 } // fim da classe Account

```

Figura B.9 Classe Account com um construtor para validar e inicializar a variável de instância balance de tipo double.

O método `credit` (linhas 19 a 22) *não* retorna nenhum dado ao completar sua tarefa, de modo que seu tipo de retorno é `void`. O método recebe um único parâmetro, chamado `amount` – um valor `double` que será somado ao saldo. A linha 21 soma `amount` ao valor atual de `balance` e atribui o resultado a `balance` (substituindo assim o valor do saldo anterior).

O método `getBalance` (linhas 25 a 28) permite aos clientes da classe (isto é, outras classes que utilizam essa classe) obter o valor de `balance` de um objeto `Account` em particular. O método especifica o tipo de retorno `double` e uma lista de parâmetros vazia.

Mais uma vez, as instruções nas linhas 15, 21 e 27 utilizam a variável de instância `balance`, mesmo ela *não* tendo sido declarada nos métodos. Podemos usar `balance` nesses métodos porque é uma variável de instância da classe.

Classe AccountTest para usar a classe Account

A classe `AccountTest` (Fig. B.10) cria dois objetos `Account` (linhas 10 e 11) e os inicializa com 50.00 e -7.53, respectivamente. As linhas 14 a 17 geram o saldo em cada `Account` chamando o método `getBalance` de `Account`. Quando o método `getBalance` é chamado para `account1` na linha 15, o valor do saldo de `account1` é retornado a partir da linha 27 da Fig. B.9 e exibido pela instrução `System.out.printf` (Fig. B.10, linhas 14 e 15). Do mesmo modo, quando o método `getBalance` é chamado para `account2` na linha 17, o valor do saldo de `account2` é retornado a partir da linha 27 da Fig. B.9 e exibido pela instrução `System.out.printf` (Fig. B.10, linhas 16 e 17). O saldo de `account2` é 0.00,

pois o construtor garantiu que ele *não* podia começar com um valor negativo. O valor é gerado na saída por `printf` com o especificador de formato `%.2f`. O especificador de formato `%f` é usado para gerar na saída valores de tipo `float` ou `double`. O `.2` entre `%` e `f` representa o número de casas decimais (2) que devem aparecer à direita do ponto decimal no número de ponto flutuante – também conhecida como a **precisão** do número. Qualquer valor de ponto flutuante gerado com `%.2f` será arredondado para centésimos – por exemplo, 123.457 seria arredondado para 123.46, 27.333 seria arredondado para 27.33 e 123.455 seria arredondado para 123.46.

```

1 // Fig. B.10: AccountTest.java
2 // Inserindo e gerando números de ponto flutuante com objetos Account.
3 import java.util.Scanner;
4
5 public class AccountTest
6 {
7     // o método main inicia a execução de aplicativos Java
8     public static void main( String[] args )
9     {
10         Account account1 = new Account( 50.00 ); // cria objeto Account
11         Account account2 = new Account( -7.53 ); // cria objeto Account
12
13         // exibe o saldo inicial de cada objeto
14         System.out.printf( "account1 balance: $%.2f\n",
15             account1.getBalance() );
16         System.out.printf( "account2 balance: $%.2f\n\n",
17             account2.getBalance() );
18
19         // cria um objeto Scanner para obter entrada da janela de comando
20         Scanner input = new Scanner( System.in );
21         double depositAmount; // deposita o valor lido do usuário
22
23         System.out.print( "Enter deposit amount for account1: " ); // prompt
24         depositAmount = input.nextDouble(); // obtém entrada do usuário
25         System.out.printf( "\nadding $%.2f to account1 balance\n\n",
26             depositAmount );
27         account1.credit( depositAmount ); // soma ao saldo de account1
28
29         // exibe os saldos
30         System.out.printf( "account1 balance: $%.2f\n",
31             account1.getBalance() );
32         System.out.printf( "account2 balance: $%.2f\n\n",
33             account2.getBalance() );
34
35         System.out.print( "Enter deposit amount for account2: " ); // prompt
36         depositAmount = input.nextDouble(); // obtém entrada do usuário
37         System.out.printf( "\nadding $%.2f to account2 balance\n\n",
38             depositAmount );
39         account2.credit( depositAmount ); // soma ao saldo de account2
40
41         // exibe os saldos
42         System.out.printf( "account1 balance: $%.2f\n",
43             account1.getBalance() );
44         System.out.printf( "account2 balance: $%.2f\n",
45             account2.getBalance() );
46     } // fim de main
47 } // fim da classe AccountTest

```

Figura B.10 Inserindo e gerando números de ponto flutuante com objetos Account. (continua)

```

account1 balance: $50.00
account2 balance: $0.00
Enter deposit amount for account1: 25.53
adding 25.53 to account1 balance
account1 balance: $75.53
account2 balance: $0.00
Enter deposit amount for account2: 123.45
adding 123.45 to account2 balance
account1 balance: $75.53
account2 balance: $123.45

```

Figura B.10 Inserindo e gerando números de ponto flutuante com objetos Account.

A linha 21 declara a variável local `depositAmount` para armazenar cada valor de depósito digitado pelo usuário. Ao contrário da variável de instância `balance` na classe `Account`, a variável local `depositAmount` em `main` *não* é inicializada com 0.0 por padrão. Contudo, essa variável não precisa ser inicializada aqui, pois seu valor vai ser determinado pela entrada do usuário.

A linha 23 pede ao usuário para que digite um valor de depósito para `account1`. A linha 24 obtém a entrada do usuário, chamando o método `nextDouble` do objeto `Scanner input`, o qual retorna um valor `double` digitado pelo usuário. As linhas 25 e 26 exibem o valor do depósito. A linha 27 chama o método `credit` do objeto `account1` e fornece `depositAmount` como argumento do método. Quando o método é chamado, o valor do argumento é atribuído ao parâmetro `amount` (linha 19 da Fig. B.9) do método `credit` (linhas 19 a 22 da Fig. B.9); então, o método `credit` soma esse valor ao `balance` (linha 21 da Fig. B.9). As linhas 30 a 33 (Fig. B.10) geram novamente os saldos dos dois objetos `Account` para mostrar que apenas o saldo de `account1` mudou.

A linha 35 pede ao usuário para que digite um valor de depósito para `account2`. A linha 36 obtém a entrada do usuário, chamando o método `nextDouble` do objeto `Scanner input`. As linhas 37 e 38 exibem o valor do depósito. A linha 39 chama o método `credit` do objeto `account2` e fornece `depositAmount` como argumento do método; então, o método `credit` soma esse valor ao saldo. Por fim, as linhas 42 a 45 geram novamente os saldos dos dois objetos `Account` para mostrar que apenas o saldo de `account2` mudou.

B.8 Para finalizar

Neste apêndice, você aprendeu a declarar variáveis de instância de uma classe para manter dados para cada objeto da classe e a declarar métodos que operam nesses dados. Aprendeu a chamar um método para que ele execute sua tarefa e a passar informações como argumentos para métodos. Aprendeu também a diferença entre uma variável local de um método e uma variável de instância de uma classe e que somente as variáveis de instância são inicializadas automaticamente. Aprendeu ainda a utilizar o construtor de uma classe para especificar os valores iniciais para as variáveis de instância de um objeto. Por fim, você aprendeu sobre números de ponto flutuante – como armazená-los com variáveis do tipo primitivo `double`, como inseri-los com um objeto `Scanner` e como formatá-los com `printf` e com o especificador de formato `%f` para propósitos de exibição. No próximo apêndice, começamos nossa introdução às instruções de controle, as quais especificam a ordem em que as ações de um programa são executadas. Você vai utilizá-las em seus métodos para especificar como devem executar suas tarefas.

Exercícios de revisão

B.1 Preencha os espaços em branco em cada um dos seguintes enunciados:

- a) Cada declaração de classe que começa com a palavra-chave _____ deve ser armazenada em um arquivo que tenha exatamente o mesmo nome da classe e termine com a extensão `.java`.
- b) Em uma declaração de classe, a palavra-chave _____ é seguida imediatamente pelo nome da classe.
- c) A palavra-chave _____ solicita memória do sistema para armazenar um objeto e, então, chama o construtor da classe correspondente para inicializar o objeto.
- d) Cada parâmetro deve especificar um(a) _____ e um(a) _____.
- e) Por padrão, as classes compiladas no mesmo diretório são consideradas como estando no mesmo pacote, conhecido como _____.
- f) Quando cada objeto de uma classe mantém sua própria cópia de um atributo, o campo que representa o atributo também é conhecido como _____.
- g) A linguagem Java fornece dois tipos primitivos para armazenar números de ponto flutuante na memória: _____ e _____.
- h) As variáveis de tipo `double` representam números de ponto flutuante _____.
- i) O método _____ de `Scanner` retorna um valor `double`.
- j) A palavra-chave `public` é um _____ de acesso.
- k) O tipo de retorno _____ indica que um método não vai retornar um valor.
- l) O método _____ de `Scanner` lê caracteres até encontrar um caractere de nova linha; então, retorna esses caracteres como uma `String`.
- m) A classe `String` está no pacote _____.
- n) Uma _____ não é exigida se você sempre se referir a uma classe com seu nome totalmente qualificado.
- o) Um _____ é um número com um ponto decimal, como 7.33, 0.0975 ou 1000.12345.
- p) As variáveis de tipo `float` representam números de ponto flutuante de _____.
- q) O especificador de formato _____ é usado para gerar na saída valores de tipo `float` ou `double`.
- r) Na linguagem Java, os tipos são divididos em duas categorias – tipos _____ e tipos _____.

B.2 Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.

- a) Por convenção, os nomes de método começam com uma letra maiúscula e todas as palavras subsequentes do nome começam com letra maiúscula.
- b) Não é exigida uma declaração `import` quando uma classe de um pacote utiliza outra do mesmo pacote.
- c) Parênteses vazios após o nome de um método em uma declaração de método indicam que o método não exige parâmetros para executar sua tarefa.
- d) Variáveis ou métodos declarados com o modificador de acesso `private` são acessíveis somente nos métodos da classe em que são declarados.
- e) Uma variável de tipo primitivo pode ser usada para chamar um método.
- f) As variáveis declaradas no corpo de um método em particular são conhecidas como variáveis de instância e podem ser usadas em todos os métodos da classe.
- g) O corpo de todo método é delimitado por chaves de abertura e fechamento (`{` e `}`).
- h) As variáveis locais de tipo primitivo são inicializadas por padrão.
- i) As variáveis de instância do tipo de referência são inicializadas com o valor `null` por padrão.

- j) Qualquer classe que contenha `public static void main( String[] args )` pode ser usada para executar um aplicativo.
 - k) O número de argumentos na chamada de método precisa corresponder ao número de parâmetros da lista de parâmetros da declaração do método.
 - l) Valores de ponto flutuante que aparecem no código-fonte são conhecidos como literais de ponto flutuante e são de tipo `float` por padrão.
- B.3** Qual é a diferença entre uma variável local e um campo?
- B.4** Explique a finalidade de um parâmetro de método. Qual é a diferença entre um parâmetro e um argumento?

Respostas dos exercícios de revisão

- B.1** a) `public`. b) `class`. c) `new`. d) tipo, nome. e) pacote padrão. f) variável de instância. g) `float`, `double`. h) precisão dupla. i) `nextDouble`. j) modificador. k) `void`. l) `nextLine`. m) `java.lang`. n) declaração `import`. o) número de ponto flutuante. p) precisão simples. q) `%f`. r) primitivos, de referência.
- B.2** a) Falsa. Por convenção, os nomes de método começam com uma letra minúscula e todas as palavras subsequentes do nome começam com letra maiúscula. b) Verdadeira. c) Verdadeira. d) Verdadeira. e) Falsa. Uma variável de tipo primitivo não pode ser usada para chamar um método – uma referência para um objeto é necessária para chamar os métodos do objeto. f) Falsa. Tais variáveis são denominadas locais e só podem ser usadas no método em que são declaradas. g) Verdadeira. h) Falsa. As variáveis de instância de tipo primitivo são inicializadas por padrão. Cada variável local deve receber um valor explicitamente. i) Verdadeira. j) Verdadeira. k) Verdadeira. l) Falsa. Tais literais são de tipo `double` por padrão.
- B.3** Uma variável local é declarada no corpo de um método e só pode ser usada do ponto em que é declarada até o final da declaração de método. Um campo é declarado em uma classe, mas não no corpo de qualquer um dos métodos da classe. Além disso, os campos são acessíveis para todos os métodos da classe. (Vamos ver uma exceção a isso no Apêndice F.)
- B.4** Um parâmetro representa informação adicional exigida por um método para executar sua tarefa. Cada parâmetro exigido por um método é especificado na declaração do método. Argumento é o valor real do parâmetro de um método. Quando um método é chamado, os valores do argumento são passados para os parâmetros correspondentes do método para que ele possa executar sua tarefa.

Exercícios

- B.5** (*Palavra-chave new*) Qual é a finalidade da palavra-chave `new`? Explique o que acontece quando você a utiliza.
- B.6** (*Construtores padrão*) O que é um construtor padrão? Como as variáveis de instância de um objeto são inicializadas se uma classe tem apenas um construtor padrão?
- B.7** (*Variáveis de instância*) Explique a finalidade de uma variável de instância.
- B.8** (*Usando classes sem importá-las*) A maioria das classes precisa ser importada antes que possam ser usadas em um aplicativo. Por que todo aplicativo pode usar as classes `System` e `String` sem antes importá-las?
- B.9** (*Usando uma classe sem importá-la*) Explique como um programa poderia usar a classe `Scanner` sem importá-la.

B.10 (Métodos set e get) Explique por que uma classe poderia fornecer um método *set* e um método *get* para uma variável de instância.

B.11 (Classe GradeBook modificada) Modifique a classe `GradeBook` (Fig. B.7) como segue:

- Inclua uma variável de instância de tipo `String` que represente o nome do professor do curso.
- Forneça um método *set* para mudar o nome do professor e um método *get* para recuperá-lo.
- Modifique o construtor para especificar dois parâmetros – um para o nome do curso e um para o nome do professor.
- Modifique o método `displayMessage` para gerar na saída a mensagem de boas-vindas e o nome do curso, seguidos de "This course is presented by: " e o nome do professor.

Use sua classe modificada em um aplicativo de teste que demonstre os novos recursos da classe.

B.12 (Classe Account modificada) Modifique a classe `Account` (Fig. B.9) para fornecer um método chamado *debit* a fim de sacar dinheiro de uma conta. Certifique-se de que o valor do débito não ultrapasse o saldo da conta. Se ultrapassar, o saldo deve ficar inalterado e o método deve imprimir uma mensagem indicando "Debit amount exceeded account balance". Modifique a classe `AccountTest` (Fig. B.10) para testar o método *debit*.

B.13 (Classe Invoice) Crie uma classe chamada `Invoice` que uma loja de ferragens possa usar para representar a fatura de um item vendido. A fatura deve incluir quatro informações como variáveis de instância – o número da peça (tipo `String`), a descrição da peça (tipo `String`), a quantidade do item que está sendo comprado (tipo `int`) e o preço por item (`double`). Sua classe deve ter um construtor que inicialize as quatro variáveis de instância. Forneça um método *set* e um método *get* para cada variável de instância. Além disso, forneça um método chamado `getInvoiceAmount` que calcule o valor da fatura (isto é, multiplique a quantidade pelo preço de cada item) e, então, retorne a quantia como um valor `double`. Se a quantidade não for positiva, deve ser configurada como 0. Se o preço por item não for positivo, deve ser configurado como 0.0. Escreva um aplicativo de teste, chamado `InvoiceTest`, que demonstre os recursos da classe `Invoice`.

B.14 (Classe Employee) Crie uma classe chamada `Employee` que inclua três variáveis de instância – um nome (tipo `String`), um sobrenome (tipo `String`) e um salário mensal (`double`). Forneça um construtor que inicialize as três variáveis de instância. Forneça um método *set* e um método *get* para cada variável de instância. Se o salário mensal não for positivo, não configure seu valor. Escreva um aplicativo de teste, chamado `EmployeeTest`, que demonstre os recursos da classe `Employee`. Crie dois objetos `Employee` e exiba o salário *anual* de cada objeto. Então, dê a cada objeto `Employee` um aumento de 10% e exiba novamente o salário anual de cada objeto.

B.15 (Classe Date) Crie uma classe chamada `Date` que inclua três variáveis de instância – um mês (tipo `int`), um dia (tipo `int`) e um ano (tipo `int`). Forneça um construtor que inicialize as três variáveis de instância e presuma que os valores fornecidos estão corretos. Forneça um método *set* e um método *get* para cada variável de instância. Forneça um método `displayDate` que exiba o mês, o dia e o ano separados por barras normais (/). Escreva um aplicativo de teste, chamado `DateTest`, que demonstre os recursos da classe `Date`.

Instruções de controle

C



Objetivos

Neste capítulo, você vai:

- Aprender técnicas básicas para a solução de problemas.
- Desenvolver algoritmos através do processo de refinamento gradual, de cima para baixo.
- Usar as instruções de seleção `if` e `if...else` para fazer uma escolha dentre ações alternativas.
- Usar a instrução de repetição `while` para executar instruções de um programa repetidamente.
- Usar repetição controlada por contador e repetição controlada por sentinela.
- Usar os operadores de atribuição compostos, de incremento e de decremento.
- Aprender os fundamentos da repetição controlada por contador.
- Usar as instruções de repetição `for` e `do...while` para executar instruções de um programa repetidamente.
- Implementar seleção múltipla usando a instrução `switch`.
- Usar as instruções `break` e `continue`.
- Usar os operadores lógicos em expressões condicionais.

Resumo

- | | |
|---|--|
| C.1 Introdução | C.12 Operadores de incremento e decremento |
| C.2 Algoritmos | C.13 Tipos primitivos |
| C.3 Pseudocódigo | C.14 Fundamentos da repetição controlada por contador |
| C.4 Estruturas de controle | C.15 Instrução de repetição for |
| C.5 Instrução de seleção simples if | C.16 Exemplos de uso da instrução for |
| C.6 Instrução de seleção dupla if...else | C.17 Instrução de repetição do...while |
| C.7 Instrução de repetição while | C.18 Instrução de seleção múltipla switch |
| C.8 Estudo de caso: repetição controlada por contador | C.19 Instruções break e continue |
| C.9 Estudo de caso: repetição controlada por sentinela | C.20 Operadores lógicos |
| C.10 Estudo de caso: instruções de controle aninhadas | C.21 Para finalizar |
| C.11 Operadores de atribuição compostos | |

Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios

C.1 Introdução

Neste apêndice, discutimos a teoria e os princípios da programação estruturada. Os conceitos apresentados aqui são fundamentais para a construção de classes e a manipulação de objetos. Apresentamos a atribuição composta da linguagem Java, os operadores de incremento e decremento e discutimos a portabilidade dos tipos primitivos. Demonstramos as instruções for, do...while e switch da linguagem Java. Por meio de uma série de exemplos curtos usando while e for, exploramos os fundamentos da repetição controlada por contador. Criamos uma versão da classe GradeBook que utiliza uma instrução switch para contar o número de notas A, B, C, D e F equivalentes a um conjunto de notas numéricas digitadas pelo usuário. Apresentamos as instruções de controle de programa break e continue. Discutimos os operadores lógicos da linguagem Java, os quais permitem usar expressões condicionais mais complexas em instruções de controle.

C.2 Algoritmos

Qualquer problema de computação pode ser resolvido pela execução de uma série de ações em uma ordem específica. Um procedimento para resolver um problema em termos

1. das **ações** a serem executadas e
2. da **ordem** na qual essas ações são executadas

é chamado de **algoritmo**. É importante especificar corretamente a ordem na qual as ações são executadas.

Considere o “algoritmo do acordar e levantar (*Rise-and-shine algorithm*)” seguido por um executivo para levantar da cama e ir para o trabalho: (1) levantar da cama; (2) tirar o pijama; (3) tomar banho; (4) vestir-se; (5) tomar o café da manhã; (6) carona solidária para o trabalho. Essa rotina possibilita que o executivo se dirija ao trabalho bem preparado para tomar decisões importantes. Suponha que as mesmas etapas sejam executadas em uma ordem ligeiramente diferente: (1) levantar da cama; (2) tirar o pijama; (3) vestir-se; (4) tomar banho; (5) tomar o café da manhã; (6) carona solidária para o trabalho. Nesse caso, nosso executivo apareceria no trabalho ensopado. A especificação da ordem na qual as instruções (ações) são executadas em um programa é chamado de **controle do progra-**

ma. Este apêndice investiga o controle do programa usando as **instruções de controle** da linguagem Java.

C.3 Pseudocódigo

Pseudocódigo é uma linguagem informal que o ajuda a desenvolver algoritmos sem ter de se preocupar com os detalhes estritos da sintaxe Java. O pseudocódigo que apresentamos é particularmente útil para o desenvolvimento de algoritmos que serão convertidos em partes estruturadas de programas em Java. O pseudocódigo é semelhante à linguagem coloquial – é conveniente e de fácil utilização, mas não é uma linguagem de programação de computador real.

Ele não é executado em computadores. Em vez disso, o ajuda a “analisar” um programa antes de tentar escrevê-lo em uma linguagem de programação, como Java. Normalmente, ele descreve apenas instruções representando as ações que ocorrem depois de você converter um programa de pseudocódigo para Java e o programa ser executado em um computador. Tais ações podem incluir entrada, saída ou cálculos.

C.4 Estruturas de controle

Normalmente, as instruções de um programa são executadas uma após a outra, na ordem em que são escritas. Esse processo é chamado de **execução sequencial**. Várias instruções Java, as quais vamos discutir em breve, permitem especificar que a próxima instrução a ser executada *não* é necessariamente a *seguinte* na sequência. Isso é chamado de **transferência de controle**.

Durante os anos 1960, tornou-se claro que o uso indiscriminado de transferências de controle era a raiz de grande parte da dificuldade encontrada pelos grupos de desenvolvimento de software. A culpa recaiu sobre a **instrução goto** (usada na maioria das linguagens de programação da época), a qual permite especificar uma transferência de controle para um dos muitos destinos em um programa. O termo **programação estruturada** tornou-se quase sinônimo da “eliminação de **goto**”. [Obs.: a linguagem Java *não* tem uma instrução goto; contudo, a palavra goto é *reservada* no Java e *não* deve ser usada como identificador em programas.]

A pesquisa demonstrou que poderiam ser escritos programas *sem* instruções goto. O desafio da época para os programadores era mudar seus estilos de “programação sem goto”. Somente nos anos 1970 é que a maioria dos programadores começou a levar a programação estruturada a sério. Os resultados foram impressionantes. O segredo do sucesso foi que os programas estruturados eram mais claros, mais fáceis de depurar e modificar, e podiam ter menos erros.

Os pesquisadores demonstraram que todos os programas podiam ser escritos em termos de apenas três estruturas de controle – a **estrutura de sequência**, a **estrutura de seleção** e a **estrutura de repetição**. Quando apresentarmos as implementações de estrutura de controle do Java, vamos nos referir a elas na terminologia da *Java Language Specification* como “instruções de controle”.

Estrutura de sequência em Java

A estrutura de sequência é incorporada à linguagem Java. A não ser que seja instruído ao contrário, o computador executa as instruções Java uma após a outra, na ordem em que são escritas – isto é, em sequência. A linguagem Java permite tantas ações quantas você desejar em uma estrutura de sequência. Conforme veremos em breve, em qualquer lugar onde possa ser colocada uma ação, podemos colocar várias delas em sequência.

Instruções de seleção em Java

A linguagem Java tem três tipos de **instruções de seleção**. A instrução `if` executa (seleciona) uma ação se uma condição for verdadeira, ou a pula caso a condição seja falsa. A instrução `if...else` executa uma ação se uma condição for verdadeira e, caso a condição seja falsa, executa uma ação diferente. A instrução `switch` executa uma de muitas ações diferentes, dependendo do valor de uma expressão.

A instrução `if` é uma **instrução de seleção simples**, pois seleciona ou ignora uma *única* ação (ou, conforme veremos em breve, um *único grupo de ações*). A instrução `if...else` é denominada **instrução de seleção dupla**, pois escolhe entre *duas ações diferentes* (ou *grupos de ações*). A instrução `switch` é denominada **instrução de seleção múltipla**, pois escolhe entre *muitas ações diferentes* (ou *grupos de ações*).

Instruções de repetição em Java

A linguagem Java fornece três **instruções de repetição** (também chamadas de **instruções de loop**) que permitem aos programas executar instruções repetidamente, desde que uma condição (denominada **condição de continuação do loop**) permaneça verdadeira. As instruções de repetição são: `while`, `do...while` e `for`. As instruções `while` e `for` executam zero ou mais vezes a ação (ou grupo de ações) que está em seus corpos – se a condição de continuação do loop for inicialmente falsa, a ação (ou grupo de ações) não será executada. A instrução `do...while` executa a ação (ou grupo de ações) que está em seu corpo *uma ou mais* vezes. As palavras `if`, `else`, `switch`, `while`, `do` e `for` são palavras-chave do Java.

C.5 Instrução de seleção simples `if`

Os programas usam instruções de seleção para escolher rumos alternativos. Por exemplo, suponha que a nota para passar em um exame seja 60. A instrução em pseudocódigo

*Se a nota do aluno for maior ou igual a 60
Imprime “Passou”*

determina se a condição “a nota do aluno é maior ou igual a 60” é verdadeira. Se for, “Passou” é impresso e a próxima instrução em pseudocódigo na ordem é “executada”. Se a condição é falsa, a instrução *Imprime* é ignorada e a próxima instrução em pseudocódigo na ordem é executada. A instrução *se* no pseudocódigo anterior pode ser facilmente convertida para a instrução Java

```
if ( studentGrade >= 60 )
    System.out.println( "Passed" );
```

C.6 Instrução de seleção dupla `if...else`

A instrução de seleção simples `if` executa a ação indicada somente quando a condição é verdadeira; caso contrário, a ação é pulada. A **instrução de seleção dupla** `if...else` permite especificar uma ação a ser executada quando a condição for verdadeira e uma ação diferente quando a condição for falsa. Por exemplo, a instrução em pseudocódigo

*Se a nota do aluno for maior ou igual a 60
Imprime “Passou”
Senão
Imprime “Reprovado”*

imprime “Passou” se a nota do aluno for maior ou igual a 60, mas imprime “Reprovado” se for menor que 60. Em um ou outro caso, após ocorrer a impressão, a próxima instrução em pseudocódigo na sequência é “executada”.

A instrução em pseudocódigo *Se...Senão* anterior pode ser escrita em Java como

```
if ( grade >= 60 )
    System.out.println( "Passed" );
else
    System.out.println( "Failed" );
```

Operador condicional (?:)

A linguagem Java fornece o **operador condicional** (?:), que pode ser usado em lugar de uma instrução `if...else`. Esse é o único **operador ternário** (operador que recebe três operandos) da linguagem Java. Juntos, os operandos e o símbolo ?: formam uma **expressão condicional**. O primeiro operando (à esquerda do ?) é uma **expressão booleana** (isto é, uma condição que é avaliada com um valor booleano – `true` ou `false`), o segundo operando (entre ? e :) é o valor da expressão condicional se a expressão booleana é verdadeira, e o terceiro operando (à direita do :) é o valor da expressão condicional se a expressão booleana for avaliada como `false`. Por exemplo, a instrução

```
System.out.println( studentGrade >= 60 ? "Passed" : "Failed" );
```

imprime o valor do argumento da expressão condicional de `println`. A expressão condicional nessa instrução é avaliada como a string “Passed” se a expressão booleana `studentGrade >= 60` for `true` e como a string “Failed” se for `false`. Assim, essa instrução com o operador condicional executa basicamente a mesma função da instrução `if...else` mostrada anteriormente nesta seção. A precedência do operador condicional é baixa; portanto, a expressão condicional inteira normalmente é colocada entre parênteses.

Instruções *if...else* aninhadas

Um programa pode testar vários casos colocando instruções `if...else` dentro de outras instruções `if...else` para criar **instruções *if...else* aninhadas**. Por exemplo, o pseudocódigo a seguir representa um `if...else` aninhado que imprime A para notas de exame maiores ou iguais a 90, B para notas de 80 a 89, C para notas de 70 a 79, D para notas de 60 a 69 e F para todas as outras notas:

```
Se a nota do aluno é maior ou igual a 90
    Imprime “A”
senão
    Se a nota do aluno é maior ou igual a 80
        Imprime “B”
    senão
        Se a nota do aluno é maior ou igual a 70
            Imprime “C”
        senão
            Se a nota do aluno é maior ou igual a 60
                Imprime “D”
            senão
                Imprime “F”
```

Esse pseudocódigo pode ser escrito em Java como

```
if ( studentGrade >= 90 )
    System.out.println( "A" );
else
    if ( studentGrade >= 80 )
        System.out.println( "B" );
    else
        if ( studentGrade >= 70 )
            System.out.println( "C" );
        else
            if ( studentGrade >= 60 )
                System.out.println( "D" );
            else
                System.out.println( "F" );
```

Se a variável `studentGrade` for maior ou igual a 90, as quatro primeiras condições na instrução `if...else` aninhada serão verdadeiras, mas somente a instrução na parte `if` da primeira instrução `if...else` será executada. Depois da execução dessa instrução, a parte `else` da instrução `if...else` “mais externa” é pulada. Muitos programadores preferem escrever a instrução `if...else` aninhada anterior como

```
if ( studentGrade >= 90 )
    System.out.println( "A" );
else if ( studentGrade >= 80 )
    System.out.println( "B" );
else if ( studentGrade >= 70 )
    System.out.println( "C" );
else if ( studentGrade >= 60 )
    System.out.println( "D" );
else
    System.out.println( "F" );
```

As duas formas são idênticas, exceto quanto ao espaçamento e o recuo, os quais o compilador ignora. A última forma evita o recuo profundo do código à direita.

Blocos

Normalmente, a instrução `if` espera somente uma instrução em seu corpo. Para incluir várias instruções no corpo de um `if` (ou no corpo de um `else` para uma instrução `if...else`), coloque-as entre chaves. Instruções contidas entre duas chaves formam um **bloco**. Um bloco pode ser colocado em qualquer lugar em que uma instrução simples possa ser colocada em um programa. O exemplo a seguir inclui um bloco na parte `else` de uma instrução `if...else`:

```
if ( grade >= 60 )
    System.out.println( "Passed" );
else
{
    System.out.println( "Failed" );
    System.out.println( "You must take this course again." );
}
```

Nesse caso, se `grade` é menor que 60, o programa executa *as duas* instruções do corpo do `else` e imprime

```
Failed
You must take this course again.
```

Observe as chaves circundando as duas instruções na cláusula `else`. Essas chaves são importantes. Sem elas, a instrução

```
System.out.println( "You must take this course again." );
```

estaria fora do corpo da parte `else` da instrução `if...else` e seria executada *independentemente* de a nota ter sido menor que 60.

Os erros de sintaxe (por exemplo, quando uma chave em um bloco é omitida do programa) são capturados pelo compilador. Um **erro de lógica** (por exemplo, quando as duas chaves em um bloco são omitidas do programa) tem seu efeito no momento da execução. Um **erro de lógica fatal** faz o programa falhar e terminar prematuramente. Um **erro de lógica não fatal** permite que o programa continue a ser executado, mas produzindo resultados incorretos.

C.7 Instrução de repetição `while`

Como exemplo de **instrução de repetição** do Java, considere um segmento de programa que encontra a primeira potência de 3 maior que 100. Suponha que a variável `int product` seja inicializada com 3. Após a execução da instrução `while` a seguir, `product` contém o resultado:

```
while ( product <= 100 )
    product = 3 * product;
```

Quando essa instrução `while` começa a ser executada, o valor da variável `product` é 3. Cada iteração da instrução `while` multiplica `product` por 3, de modo que `product` assume os valores 9, 27, 81 e 243 sucessivamente. Quando a variável `product` se torna 243, a condição da instrução `while` — `product <= 100` — se torna falsa. Isso termina a repetição; portanto, o valor final de `product` é 243. Nesse ponto, a execução do programa continua na próxima instrução após a instrução `while`.



Erro de programação comum C.1

*Não fornecer, no corpo de uma instrução `while`, uma ação que finalmente faça a condição do `while` se tornar falsa, normalmente resulta em um erro de lógica chamado *loop infinito* (o *loop* nunca termina).*

C.8 Estudo de caso: repetição controlada por contador

Para ilustrar como os algoritmos são desenvolvidos, modificamos a classe `GradeBook` do Apêndice B para resolver duas variações de um problema que tira a média das notas dos alunos. Considere o problema a seguir:

Uma classe de 10 alunos fez um teste. As notas (valores inteiros no intervalo de 0 a 100) desse teste estão disponíveis para você. Determine a média da classe no teste.

A média da classe é igual à soma das notas dividida pelo número de alunos. O algoritmo para resolver esse problema em um computador deve inserir cada nota, monitorar o total de todas as notas inseridas, efetuar o cálculo da média e imprimir o resultado.

Algoritmo em pseudocódigo com repetição controlada por contador

Vamos usar pseudocódigo para listar as ações a serem executadas e especificar a ordem em que elas devem ser executadas. Usamos **repetição controlada por contador** para inserir

as notas uma por vez. Essa técnica utiliza uma variável chamada **contadora** (ou **variável de controle**) para controlar o número de vezes que um conjunto de instruções será executado. Neste exemplo, a repetição termina quando o contador passa de 10. Esta seção apresenta um algoritmo em pseudocódigo totalmente desenvolvido (Fig. C.1) e uma versão da classe `GradeBook` (Fig. C.2) que implementa o algoritmo em um método Java. Em seguida, apresentamos um aplicativo (Fig. C.3) que demonstra o algoritmo em ação.

Observe no algoritmo da Fig. C.1 as referências a um total e a um contador. O **total** é uma variável usada para acumular a soma de vários valores. O contador é uma variável usada para contar – neste caso, o contador de notas indica qual das 10 notas está para ser inserida pelo usuário. As variáveis usadas para armazenar totais normalmente são inicializadas com zero antes de serem usadas em um programa.

```

1  Configura o total com zero
2  Configura o contador de notas com um
3
4  Enquanto o contador de notas é menor ou igual a dez
5      Pede ao usuário para que insira a próxima nota
6      Insere a próxima nota
7      Soma a nota ao total
8      Soma um ao contador de notas
9
10 Configura a média da classe com o total dividido por dez
11 Imprime a média da classe

```

Figura C.1 Algoritmo em pseudocódigo que usa repetição controlada por contador para resolver o problema da média da classe.

Implementando repetição controlada por contador na classe `GradeBook`

A classe `GradeBook` (Fig. C.2) contém um construtor (linhas 11 a 14) que atribui um valor à variável de instância `courseName` da classe (declarada na linha 8). As linhas 17 a 20, 23 a 26 e 29 a 34 declaram os métodos `setCourseName`, `getCourseName` e `displayMessage`, respectivamente. As linhas 37 a 66 declaram o método `determineClassAverage`, o qual implementa o algoritmo de média da classe descrito pelo pseudocódigo da Fig. C.1.

A linha 40 declara e inicializa a variável `Scanner input`, a qual é usada para ler os valores inseridos pelo usuário. As linhas 42 a 45 declaram as variáveis locais `total`, `gradeCounter`, `grade` e `average` como sendo de tipo `int`. A variável `grade` armazena a entrada do usuário.

```

1  // Fig. C.2: GradeBook.java
2  // Classe GradeBook que resolve o problema da média da classe usando
3  // repetição controlada por contador.
4  import java.util.Scanner; // o programa usa a classe Scanner
5
6  public class GradeBook
7  {
8      private String courseName; // nome do curso representado por GradeBook
9

```

Figura C.2 Classe `GradeBook` que resolve o problema da média da classe usando repetição controlada por contador.

```

10 // o construtor inicializa courseName
11 public GradeBook( String name )
12 {
13     courseName = name; // inicializa courseName
14 } // fim do construtor
15
16 // método para configurar o nome do curso
17 public void setCourseName( String name )
18 {
19     courseName = name; // armazena o nome do curso
20 } // fim do método setCourseName
21
22 // método para recuperar o nome do curso
23 public String getCourseName()
24 {
25     return courseName;
26 } // fim do método getCourseName
27
28 // exibe uma mensagem de boas-vindas ao usuário do aplicativo de notas
29 public void displayMessage()
30 {
31     // getCourseName recebe o nome do curso
32     System.out.printf( "Welcome to the grade book for\n%s!\n\n",
33         getCourseName() );
34 } // fim do método displayMessage
35
36 // determina a média da classe com base nas 10 notas digitadas pelo usuário
37 public void determineClassAverage()
38 {
39     // cria objeto Scanner para obter a entrada da janela de comando
40     Scanner input = new Scanner( System.in );
41
42     int total; // soma das notas digitadas pelo usuário
43     int gradeCounter; // número de notas a serem inseridas a seguir
44     int grade; // valor da nota digitada pelo usuário
45     int average; // média das notas
46
47     // fase de inicialização
48     total = 0; // inicializa o total
49     gradeCounter = 1; // inicializa o contador do loop
50
51     // a fase de processamento usa repetição controlada por contador
52     while ( gradeCounter <= 10 ) // faz loop 10 vezes
53     {
54         System.out.print( "Enter grade: " ); // prompt
55         grade = input.nextInt(); // insere a próxima nota
56         total = total + grade; // soma a nota ao total
57         gradeCounter = gradeCounter + 1; // incrementa o contador por 1
58     } // fim do while
59
60     // fase de conclusão
61     average = total / 10; // a divisão inteira produz um resultado inteiro
62
63     // exibe o total e a média das notas
64     System.out.printf( "\nTotal of all 10 grades is %d\n", total );
65     System.out.printf( "Class average is %d\n", average );
66 } // fim do método determineClassAverage
67 } // fim da classe GradeBook

```

Figura C.2 Classe GradeBook que resolve o problema da média da classe usando repetição controlada por contador.

As declarações (nas linhas 42 a 45) aparecem no corpo do método `determineClassAverage`. A declaração de uma variável local deve aparecer *antes* que a variável seja usada nesse método. Uma variável local não pode ser acessada fora do método em que é declarada.

As atribuições (nas linhas 48 e 49) inicializam `total` com 0 e `gradeCounter` com 1. A linha 52 indica que a instrução `while` deve continuar o loop (também chamado de **iteração**) enquanto o valor de `gradeCounter` for menor ou igual a 10. Enquanto essa condição permanece verdadeira, a instrução `while` executa repetidamente as instruções que estão entre as chaves que delimitam seu corpo (linhas 54 a 57).

A linha 54 exibe o prompt "Enter grade: ". A linha 55 lê a nota inserida pelo usuário e a atribui à variável `grade`. Então, a linha 56 soma a nova nota inserida pelo usuário ao `total` e atribui o resultado a `total`, o que substitui seu valor anterior.

A linha 57 soma 1 a `gradeCounter` para indicar que o programa processou uma nota e está pronto para inserir a próxima nota do usuário. Incrementar `gradeCounter` faz com que finalmente ultrapasse 10. Então, o loop termina, pois sua condição (linha 52) se torna falsa.

Quando o loop termina, a linha 61 faz o cálculo da média e atribui o resultado à variável `average`. A linha 64 usa o método `printf` de `System.out` para exibir o texto "Total of all 10 grades is ", seguido do valor da variável `total`. Então, a linha 65 usa `printf` para exibir o texto "Class average is ", seguido do valor da variável `average`. Após atingir a linha 66, o método `determineClassAverage` retorna o controle para o método chamador (isto é, `main` em `GradeBookTest` da Fig. C.3).

Classe *GradeBookTest*

A classe `GradeBookTest` (Fig. C.3) cria um objeto da classe `GradeBook` (Fig. C.2) e demonstra seus recursos. As linhas 10 e 11 da Fig. C.3 criam um novo objeto `GradeBook` e o atribuem à variável `myGradeBook`. A `String` da linha 11 é passada para o construtor de `GradeBook` (linhas 11 a 14 da Fig. C.2). A linha 13 chama o método `displayMessage` de `myGradeBook` para exibir uma mensagem de boas-vindas para o usuário. Então, a linha 14 chama o método `determineClassAverage` de `myGradeBook` para permitir que o usuário insira 10 notas, para as quais o método calcula e imprime a média – o método executa o algoritmo mostrado na Fig. C.1.

```

1 // Fig. C.3: GradeBookTest.java
2 // Cria objeto GradeBook e chama seu método determineClassAverage.
3
4 public class GradeBookTest
5 {
6     public static void main( String[] args )
7     {
8         // cria objeto myGradeBook de GradeBook e
9         // passa o nome do curso para o construtor
10        GradeBook myGradeBook = new GradeBook(
11            "CS101 Introduction to Java Programming" );
12
13        myGradeBook.displayMessage(); // exibe a mensagem de boas-vindas
14        myGradeBook.determineClassAverage(); // acha a média das 10 notas
15    } // fim de main
16 } // fim da classe GradeBookTest

```

Figura C.3 A classe `GradeBookTest` cria um objeto da classe `GradeBook` (Fig. C.2) e chama seu método `determineClassAverage`. (continua)

```
Welcome to the grade book for  
CS101 Introduction to Java Programming!
```

```
Enter grade: 67  
Enter grade: 78  
Enter grade: 89  
Enter grade: 67  
Enter grade: 87  
Enter grade: 98  
Enter grade: 93  
Enter grade: 85  
Enter grade: 82  
Enter grade: 100
```

```
Total of all 10 grades is 846  
Class average is 84
```

Figura C.3 A classe GradeBookTest cria um objeto da classe GradeBook (Fig. C.2) e chama seu método determineClassAverage.

Observações sobre divisão inteira e truncamento

O cálculo da média efetuado pelo método determineClassAverage em resposta à chamada de método na linha 14 da Fig. C.3 produz um resultado inteiro. A saída do programa indica que a soma dos valores de nota no exemplo de execução é 846, a qual, quando dividida por 10, deve produzir o número de ponto flutuante 84.6. Contudo, o resultado do cálculo total / 10 (linha 61 da Fig. C.2) é o valor inteiro 84, pois total e 10 são ambos inteiros. Dividir dois valores inteiros resulta na **divisão inteira** – qualquer parte fracionária do cálculo é perdida (isto é, **truncada**).

C.9 Estudo de caso: repetição controlada por sentinela

Vamos generalizar o problema da média da classe da seção C.8. Considere o problema a seguir:

Desenvolva um programa para tirar a média da classe que processe as notas de um número arbitrário de alunos cada vez que for executado.

No exemplo da média da classe anterior, o enunciado do problema especificava o número de alunos, de modo que o número de notas (10) era conhecido antecipadamente. Neste exemplo, não é dada uma indicação sobre quantas notas o usuário digitará durante a execução do programa. O programa deve processar um número arbitrário de notas. Como ele pode saber quando deve parar de ler notas do usuário? Como vai saber quando deve calcular e imprimir a média da classe?

Um modo de resolver esse problema é usar um valor especial chamado **sentinela** (também chamado de **valor de sinal**, **valor dummy** ou **valor flag**) para indicar o “fim da entrada de dados”. O usuário digita notas até que todas as que forem legítimas tenham sido introduzidas. Então, ele digita o valor de sentinela para indicar que mais nenhuma nota vai ser inserida. A **repetição controlada por sentinela** é muitas vezes chamada de **repetição indefinida**, porque o número de repetições *não* é conhecido antes que o loop comece a ser executado.

Claramente, deve ser escolhido um valor de sentinela que não possa ser confundido com um valor de entrada aceitável. As notas de um teste são valores inteiros não negativos; portanto, -1 é um valor de sentinela aceitável para este problema. Assim, uma execução do programa da média da classe poderia processar um fluxo de entradas como 95, 96, 75, 74, 89 e -1. Então, o programa calcularia e imprimiria a média da classe para as notas 95, 96, 75, 74 e 89; como -1 é o valor de sentinela, *não* deve entrar no cálculo da média. O pseudocódigo completo para o problema da média da classe está mostrado na Fig. C.4.

```

1  Inicializa o total com zero
2  Inicializa o contador com zero
3
4  Pede ao usuário para que insira a primeira nota
5  Insere a primeira nota (possivelmente a sentinela)
6
7  Enquanto o usuário não tiver digitado a sentinela
8      Soma essa nota ao total parcial
9      Soma um ao contador de notas
10     Pede ao usuário para que insira a próxima nota
11     Insere a próxima nota (possivelmente a sentinela)
12
13 Se o contador não for igual a zero
14     Configura a média com o total dividido pelo contador
15     Imprime a média
16 senão
17     Imprime "No grades were entered"
```

Figura C.4 Algoritmo em pseudocódigo do problema da média da classe com repetição controlada por sentinela.

Implementando repetição controlada por sentinela na classe GradeBook

A Figura C.5 mostra a classe Java GradeBook contendo o método determineClassAverage que implementa o algoritmo em pseudocódigo da Fig. C.4. Embora cada nota seja um valor inteiro, o cálculo da média provavelmente vai produzir um número com um ponto decimal – em outras palavras, um número real (isto é, de ponto flutuante). O tipo int não pode representar esse número, de modo que essa classe utiliza o tipo double para isso.

```

1  // Fig. C.5: GradeBook.java
2  // Classe GradeBook que resolve o problema da média da classe usando
3  // repetição controlada por sentinela.
4  import java.util.Scanner; // o programa usa a classe Scanner
5
6  public class GradeBook
7  {
8      private String courseName; // nome do curso representado por GradeBook
9
10     // o construtor inicializa courseName
11     public GradeBook( String name )
12     {
```

Figura C.5 Classe GradeBook que resolve o problema da média da classe usando repetição controlada por sentinela.

```

13     courseName = name; // inicializa courseName
14 } // fim do construtor
15
16 // método para configurar o nome do curso
17 public void setCourseName( String name )
18 {
19     courseName = name; // armazena o nome do curso
20 } // fim do método setCourseName
21
22 // método para recuperar o nome do curso
23 public String getCourseName()
24 {
25     return courseName;
26 } // fim do método getCourseName
27
28 // exibe uma mensagem de boas-vindas ao usuário do aplicativo de notas
29 public void displayMessage()
30 {
31     // getCourseName recebe o nome do curso
32     System.out.printf( "Welcome to the grade book for\n%s!\n\n",
33         getCourseName() );
34 } // fim do método displayMessage
35
36 // determina a média de um número arbitrário de notas
37 public void determineClassAverage()
38 {
39     // cria objeto Scanner para obter a entrada da janela de comando
40     Scanner input = new Scanner( System.in );
41
42     int total; // soma das notas
43     int gradeCounter; // número de notas inseridas
44     int grade; // valor da nota
45     double average; // número com ponto decimal para a média
46
47     // fase de inicialização
48     total = 0; // inicializa o total
49     gradeCounter = 0; // inicializa o contador do loop
50
51     // fase de processamento
52     // solicita a entrada e lê a nota do usuário
53     System.out.print( "Enter grade or -1 to quit: " );
54     grade = input.nextInt();
55
56     // faz loop até que o valor de sentinela seja lido do usuário
57     while ( grade != -1 )
58     {
59         total = total + grade; // soma a nota ao total
60         gradeCounter = gradeCounter + 1; // incrementa o contador
61
62         // solicita a entrada e lê a próxima nota do usuário
63         System.out.print( "Enter grade or -1 to quit: " );
64         grade = input.nextInt();
65     } // fim do while
66
67     // fase de conclusão
68     // se o usuário digitou pelo menos uma nota...
69     if ( gradeCounter != 0 )
70     {
71         // calcula a média de todas as notas digitadas

```

Figura C.5 Classe GradeBook que resolve o problema da média da classe usando repetição controlada por sentinela. (continua)

```

72         average = (double) total / gradeCounter;
73
74         // exibe o total e a média (com dois dígitos de precisão)
75         System.out.printf( "\nTotal of the %d grades entered is %d\n",
76                             gradeCounter, total );
77         System.out.printf( "Class average is %.2f\n", average );
78     } // fim do if
79     else // nenhuma nota foi inserida; portanto, gera a mensagem apropriada na saída
80         System.out.println( "No grades were entered" );
81 } // fim do método determineClassAverage
82 } // fim da classe GradeBook

```

Figura C.5 Classe GradeBook que resolve o problema da média da classe usando repetição controlada por sentinela.

Neste exemplo, vemos que as instruções de controle podem ser *empilhadas* umas sobre as outras (em sequência). A instrução `while` (linhas 57 a 65) é seguida na sequência por uma instrução `if...else` (linhas 69 a 80). Grande parte do código desse programa é idêntica ao da Fig. C.2; portanto, vamos nos concentrar nos conceitos novos.

A linha 45 declara a variável `double average`, a qual nos permite armazenar a média da classe como um número de ponto flutuante. A linha 49 inicializa `gradeCounter` com 0, pois nenhuma nota foi inserida ainda. Para manter um registro preciso do número de notas inseridas, o programa incrementa `gradeCounter` somente quando o usuário digita uma nota válida.

Lógica de programa para repetição controlada por sentinela versus repetição controlada por contador

Compare a lógica de programa para repetição controlada por sentinela deste aplicativo com a da repetição controlada por contador da Fig. C.2. Na repetição controlada por contador, cada iteração da instrução `while` (por exemplo, linhas 52 a 58 da Fig. C.2) lê um valor do usuário pelo número especificado de iterações. Na repetição controlada por sentinela, o programa lê o primeiro valor (linhas 53 e 54 da Fig. C.5) antes de atingir o `while`. Esse valor determina se o fluxo de controle do programa deve entrar no corpo do `while`. Se a condição do `while` é falsa, o usuário digitou o valor de sentinela, de modo que o corpo do `while` não é executado (isto é, nenhuma nota foi inserida). Por outro lado, se a condição é verdadeira, o corpo começa a ser executado e o loop soma o valor da nota ao `total` (linha 59). Então, as linhas 63 e 64 no corpo do loop inserem o próximo valor do usuário. Em seguida, o controle do programa chega à chave de fechamento do corpo do loop na linha 65, de modo que a execução continua com o teste da condição do `while` (linha 57). A condição usa a nota mais recente inserida pelo usuário para determinar se o corpo do loop deve ser executado novamente. O valor da variável `grade` é sempre inserido pelo usuário imediatamente antes de o programa testar a condição do `while`. Isso permite que o programa determine se o valor que acabou de ser inserido é o valor de sentinela *antes* de processar esse valor (isto é, somá-lo ao `total`). Se o valor de sentinela for inserido, o loop termina e o programa não soma -1 ao `total`.

Depois que o loop termina, a instrução `if...else` das linhas 69 a 80 é executada. A condição na linha 69 determina se alguma nota foi inserida. Se nenhuma foi inserida, a parte `else` (linhas 79 e 80) da instrução `if...else` é executada, exibe a mensagem "No grades were entered" e o método retorna o controle para o método chamador.

Conversão explícita e implícita entre tipos primitivos

Se pelo menos uma nota foi inserida, a linha 72 da Fig. C.5 calcula a média das notas. Lembre que na Fig. C.2 a divisão inteira produz um resultado inteiro. Mesmo a variável `average` sendo declarada como `double` (linha 45), o cálculo

```
average = total / gradeCounter;
```

perde a parte fracionária do quociente *antes* que o resultado da divisão seja atribuído a `average`. Isso ocorre porque `total` e `gradeCounter` são *ambos* valores inteiros, e a divisão inteira produz um resultado inteiro. Para efetuar cálculo de ponto flutuante com valores inteiros, devemos tratar esses valores temporariamente como números de ponto flutuante. A linguagem Java fornece o **operador de conversão unário** para realizar essa tarefa. A linha 72 usa o operador de conversão (**`double`**) – um operador unário – para criar uma cópia de ponto flutuante *temporária* de seu operando `total` (o qual aparece à direita do operador). Usar um operador de conversão dessa maneira é chamado de **conversão explícita** ou **conversão de tipo**. O valor armazenado em `total` ainda é inteiro.

Agora o cálculo consiste em um valor de ponto flutuante (a versão `double` temporária de `total`) dividido pelo valor inteiro `gradeCounter`. A linguagem Java sabe avaliar apenas expressões aritméticas nas quais os tipos dos operandos são *idênticos*. Para garantir que os operandos sejam do mesmo tipo, a linguagem Java executa uma operação chamada **promoção** (ou **conversão implícita**) nos operandos selecionados. Por exemplo, em uma expressão contendo valores dos tipos `int` e `double`, os valores `int` são promovidos para `double`. Neste exemplo, o valor de `gradeCounter` é promovido para o tipo `double` e, então, a divisão de ponto flutuante é efetuada e o resultado do cálculo é atribuído a `average`. Desde que o operador de conversão (`double`) seja aplicado a *toda* variável no cálculo, este vai gerar um resultado `double`.

O operador de conversão é formado pela colocação de parênteses em torno do nome de qualquer tipo. Ele é um **operador unário** (isto é, um operador que recebe apenas um operando). A linguagem Java também suporta versões unárias dos operadores mais (+) e menos (-), de modo que você pode escrever expressões como `-7` ou `+5`. Os operadores de conversão são associados da direita para a esquerda e têm a mesma precedência dos outros operadores unários, como o `+` unário e o `-` unário. (Consulte a tabela de precedência de operadores no Apêndice K.)

A linha 77 exibe a média da classe. Neste exemplo, exibimos a média da classe arredondada para o centésimo mais próximo. O especificador de formato `%.2f` na string de controle de formato de `printf` indica que o valor da variável `average` deve ser exibido com dois dígitos de precisão à direita do ponto decimal – indicado pelo `.2` no especificador de formato. As três notas inseridas durante o exemplo de execução da classe `GradeBookTest` (Fig. C.6) totalizam 257, o que produz a média 85.666666.... O método `printf` usa a precisão do especificador de formato para arredondar o valor para o número especificado de dígitos. Neste programa, a média é arredondada na posição dos centésimos e exibida como 85.67.

```
1 // Fig. C.6: GradeBookTest.java
2 // Cria objeto GradeBook e chama seu método determineClassAverage.
3
4 public class GradeBookTest
5 {
```

Figura C.6 A classe `GradeBookTest` cria um objeto da classe `GradeBook` (Fig. C.5) e chama seu método `determineClassAverage`. (continua)

```

6      public static void main( String[] args )
7      {
8          // cria objeto myGradeBook de GradeBook e
9          // passa o nome do curso para o construtor
10         GradeBook myGradeBook = new GradeBook(
11             "CS101 Introduction to Java Programming" );
12
13         myGradeBook.displayMessage(); // exibe a mensagem de boas-vindas
14         myGradeBook.determineClassAverage(); // acha a média das notas
15     } // fim de main
16 } // fim da classe GradeBookTest

```

```

Welcome to the grade book for
CS101 Introduction to Java Programming!

Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of the 3 grades entered is 257
Class average is 85.67

```

Figura C.6 A classe `GradeBookTest` cria um objeto da classe `GradeBook` (Fig. C.5) e chama seu método `determineClassAverage`.

C.10 Estudo de caso: instruções de controle aninhadas

Vimos que as instruções de controle podem ser empilhadas umas sobre as outras (em sequência). Neste estudo de caso, examinamos a única outra maneira estruturada pela qual instruções de controle podem ser conectadas – pelo **aninhamento** de uma instrução de controle dentro de outra. Considere o problema a seguir:

Uma faculdade oferece um curso preparatório para o exame de admissão para corretores de imóveis. No ano passado, 10 alunos que concluíram esse curso fizeram o exame. A faculdade quer saber qual foi o desempenho de seus alunos no exame. Sua incumbência é escrever um programa para resumir os resultados. Você recebeu uma listagem desses 10 alunos. Ao lado de cada nome está escrito 1 se o aluno passou no exame ou 2 se não passou.

Seu programa deve analisar os resultados do exame como segue:

1. Inserir o resultado de cada teste (isto é, 1 ou 2). Exibir a mensagem “Enter result” na tela sempre que o programa solicitar outro resultado.
2. Contar o número de resultados de cada tipo.
3. Exibir um resumo dos resultados do teste, indicando o número de alunos aprovados e o número de reprovados.
4. Se mais de oito alunos passaram no exame, imprimir a mensagem “Bonus to instructor!”

O pseudocódigo completo aparece na Fig. C.7. A classe Java que implementa o algoritmo em pseudocódigo e dois exemplos de execução aparecem na Fig. C.8. As linhas 13 a 16 de `main` declaram as variáveis utilizadas pelo método `processExamResults`

da classe `Analysis` para processar os resultados do exame. Várias dessas declarações utilizam a capacidade do Java de incorporar nelas a inicialização de variável (0 é atribuído a `passes`, 0 é atribuído a `failures` e 1 é atribuído a `studentCounter`). Os loops dos programas podem exigir inicialização no início de cada repetição – normalmente realizada por meio de instruções de atribuição, em vez de declarações. A linguagem Java exige que as variáveis locais sejam inicializadas antes que seus valores sejam usados em uma expressão.

```

1  Inicializa as aprovações com zero
2  Inicializa as reprovações com zero
3  Inicializa o contador de alunos com um
4
5  Enquanto o contador de alunos é menor ou igual a dez
6      Pede ao usuário para que insira o próximo resultado do exame
7      Insere o próximo resultado do exame
8
9      Se o aluno passou
10         Soma um às aprovações
11     Senão
12         Soma um às reprovações
13
14     Soma um ao contador de alunos
15
16 Imprime o número de aprovações
17 Imprime o número de reprovações
18
19 Se mais de oito alunos passaram
20     Imprime "Bonus to instructor!"

```

Figura C.7 Pseudocódigo para o problema dos resultados do exame.

A instrução `while` (linhas 19 a 33) faz loop 10 vezes. Durante cada iteração, o loop insere e processa um resultado do exame. Observe que a instrução `if...else` (linhas 26 a 29) para processar cada resultado está *aninhada* na instrução `while`. Se `result` é 1, a instrução `if...else` incrementa `passes`; caso contrário, presume que `result` é 2 e incrementa `failures`. A linha 32 incrementa `studentCounter` antes que a condição do loop seja testada novamente na linha 19. Depois de inseridos 10 valores, o loop termina e a linha 36 exibe o número de aprovações (`passes`) e reprovações (`failures`). A instrução `if` nas linhas 39 e 40 determina se mais de oito alunos passaram no exame e, se assim for, gera na saída a mensagem "Bonus to instructor!".

```

1  // Fig. C.8: Analysis.java
2  // Análise dos resultados do exame usando instruções de controle aninhadas.
3  import java.util.Scanner; // usa a classe Scanner
4
5  public class Analysis
6  {
7      public static void main( String[] args )
8      {
9          // cria objeto Scanner para obter a entrada da janela de comando
10         Scanner input = new Scanner( System.in );
11
12         // inicializando variáveis em declarações

```

Figura C.8 Análise dos resultados do exame usando instruções de controle aninhadas. (continua)


```

13     int passes = 0; // número de aprovações
14     int failures = 0; // número de reprovações
15     int studentCounter = 1; // contador de alunos
16     int result; // um resultado do exame (obtem o valor do usuário)
17
18     // processa 10 alunos usando loop controlado por contador
19     while ( studentCounter <= 10 )
20     {
21         // solicita a entrada do usuário e obtém o valor do usuário
22         System.out.print( "Enter result (1 = pass, 2 = fail): " );
23         result = input.nextInt();
24
25         // if...else aninhado na instrução while
26         if ( result == 1 ) // resultado de if é 1,
27             passes = passes + 1; // incrementa passes;
28         else // resultado de else não é 1; logo
29             failures = failures + 1; // incrementa failures
30
31         // incrementa studentCounter para que o loop finalmente termine
32         studentCounter = studentCounter + 1;
33     } // fim do while
34
35     // fase de conclusão; prepara e exibe os resultados
36     System.out.printf( "Passed: %d\nFailed: %d\n", passes, failures );
37
38     // determina se mais de 8 alunos passaram
39     if ( passes > 8 )
40         System.out.println( "Bonus to instructor!" );
41 } // fim de main
42 } // fim da classe Analysis

```

```

Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Bonus to instructor!

```

Figura C.8 Análise dos resultados do exame usando instruções de controle aninhadas.

Durante o exemplo de execução, a condição na linha 39 do método main é verdadeira – mais de oito alunos passaram no exame, de modo que o programa gera uma mensagem de bônus para o professor.

Este exemplo contém somente uma classe, com o método main realizando todo o trabalho da classe. Ocasionalmente, quando não fizer sentido criar uma classe *reutilizável* para demonstrar um conceito, colocaremos as instruções do programa inteiramente dentro do método main de uma única classe.

C.11 Operadores de atribuição compostos

Os **operadores de atribuição compostos** abreviam as expressões de atribuição. Instruções como

```
variável = variável operador expressão;
```

onde *operador* é um dos operadores binários +, -, *, / ou % (ou outros que discutiremos mais adiante no texto), podem ser escritas na forma

```
variável operador= expressão;
```

Por exemplo, você pode abreviar a instrução

```
c = c + 3;
```

com o **operador de atribuição composto de adição**, +=, como

```
c += 3;
```

O operador += soma o valor da expressão à sua direita ao valor da variável à sua esquerda e armazena o resultado na variável à esquerda do operador. Assim, a expressão de atribuição `c += 3` soma 3 a `c`. A Figura C.9 mostra os operadores de atribuição compostos aritméticos, exemplos de expressões usando os operadores e explicações sobre o que os operadores fazem.

Operador de atribuição	Exemplo de expressão	Explicação	Atribui
Presuma: <code>int c = 3, d = 5, e = 4, f = 6, g = 12;</code>			
+=	<code>c += 7</code>	<code>c = c + 7</code>	10 a c
-=	<code>d -= 4</code>	<code>d = d - 4</code>	1 a d
*=	<code>e *= 5</code>	<code>e = e * 5</code>	20 a e
/=	<code>f /= 3</code>	<code>f = f / 3</code>	2 a f
%=	<code>g %= 9</code>	<code>g = g % 9</code>	3 a g

Figura C.9 Operadores de atribuição compostos aritméticos.

C.12 Operadores de incremento e decremento

A linguagem Java fornece dois operadores unários (resumidos na Fig. C.10) para somar 1 ou subtrair 1 do valor de uma variável numérica. São eles: o **operador de incremento** unário, ++, e o **operador de decremento** unário, --. Um programa pode incrementar por 1 o valor de uma variável chamada `c` usando o operador de incremento, ++, em vez da expressão `c = c + 1` ou `c += 1`. Um operador de incremento ou decremento prefixado (colocado antes) a uma variável é referido como **operador de incremento prefixado** ou **operador de decremento prefixado**, respectivamente. Um operador de incremento ou decremento pós-fixado (colocado depois) a uma variável é referido como **operador de incremento pós-fixado** ou **operador de decremento pós-fixado**, respectivamente.

Operador	Nome do operador	Exemplo de expressão	Explicação
++	incremento prefixado	++a	Incrementa a por 1 e, então, usa o novo valor de a na expressão em que a está.
++	incremento pós-fixado	a++	Usa o valor atual de a na expressão em que a está e, então, incrementa a por 1.
--	decremento prefixado	--b	Decrementa b por 1 e, então, usa o novo valor de b na expressão em que b está.
--	decremento pós-fixado	b--	Usa o valor atual de b na expressão em que b está e, então, decrementa b por 1.

Figura C.10 Operadores de incremento e decremento.

Usar o operador de incremento (ou decremento) prefixado para somar 1 (ou subtrair 1) a uma variável é conhecido como **pré-incrementar** (ou **pré-decrementar**). Isso faz com que a variável seja incrementada (decrementada) por 1; então, o novo valor da variável é usado na expressão em que ela aparece. Usar o operador de incremento (ou decremento) pós-fixado para somar 1 (ou subtrair 1) a uma variável é conhecido como **pós-incrementar** (ou **pós-decrementar**). Isso faz que o valor atual da variável seja usado na expressão em que ela aparece; então, o valor da variável é incrementado (decrementado) por 1.

A Figura C.11 demonstra a diferença entre as versões de incremento prefixado e incremento pós-fixado do operador de incremento ++. O operador de decremento (--) funciona de modo semelhante. A linha 11 inicializa a variável c com 5, e a linha 12 gera o valor inicial de c. A linha 13 gera o valor da expressão c++. Essa expressão pós-incrementa a variável c; portanto, o valor original de c (5) é gerado e, então, o valor de c é incrementado (para 6). Assim, a linha 13 gera novamente o valor inicial de c (5). A linha 14 gera o novo valor de c (6) para provar que o valor da variável foi mesmo incrementado na linha 13.

```

1 // Fig. C.11: Increment.java
2 // Operadores de incremento e decremento prefixados e pós-fixados
3
4 public class Increment
5 {
6     public static void main( String[] args )
7     {
8         int c;
9
10        // demonstra o operador de incremento pós-fixado
11        c = 5; // atribui 5 a c
12        System.out.println( c ); // imprime 5
13        System.out.println( c++ ); // imprime 5 e então pós-incrementa
14        System.out.println( c ); // imprime 6
15
16        System.out.println(); // pula uma linha
17
18        // demonstra o operador de incremento prefixado
19        c = 5; // assign 5 to c
20        System.out.println( c ); // imprime 5

```

Figura C.11 Pré-incrementando e pós-incrementando. (continua)

```

21      System.out.println( ++c ); // preincrementa e então imprime 6
22      System.out.println( c ); // imprime 6
23  } // fim de main
24 } // fim da classe Increment

```

```

5
5
6

5
6
6

```

Figura C.11 Pré-incrementando e pós-incrementando.

A linha 19 reconfigura o valor de `c` com 5, e a linha 20 gera o valor de `c`. A linha 21 gera o valor da expressão `++c`. Essa expressão pré-incrementa `c`, de modo que seu valor é incrementado; então, o novo valor (6) é gerado na saída. A linha 22 gera novamente o valor de `c` na saída para mostrar que o valor de `c` ainda é 6 após a execução da linha 21.

Ao incrementar ou decrementar uma variável em uma instrução sozinha, as formas de incremento prefixado e pós-fixado têm o mesmo efeito, e as formas de decremento prefixado e pós-fixado têm o mesmo efeito. É somente quando uma variável aparece no contexto de uma expressão maior que a sua pré-incrementação e pós-incrementação têm efeitos diferentes (e o mesmo vale para pré-decrementar e pós-decrementar).

C.13 Tipos primitivos

A tabela do Apêndice L lista os oito tipos primitivos da linguagem Java. Assim como suas linguagens predecessoras C e C++, Java exige que todas as variáveis tenham um tipo. Por isso, Java é referida como uma **linguagem fortemente tipada**.

Em C e C++, os programadores frequentemente precisam escrever versões separadas dos programas a fim de oferecer suporte para diferentes plataformas de computador, pois não há garantia de que os tipos primitivos sejam idênticos de um computador para outro. Por exemplo, um valor `int` em uma máquina pode ser representado por 16 bits (2 bytes) de memória, em uma segunda máquina por 32 bits (4 bytes) de memória e em outra por 64 bits (8 bytes) de memória. Em Java, os valores `int` têm sempre 32 bits (4 bytes).



Dica de portabilidade C.1

Os tipos primitivos em Java são portáveis entre todas as plataformas de computador que suportam Java.

No Apêndice L, cada tipo é listado com seu tamanho em bits (em um byte existem oito bits) e seu intervalo de valores. Como os projetistas da linguagem Java querem garantir a portabilidade, eles usam padrões reconhecidos internacionalmente para formatos de caractere (Unicode; para obter mais informações, visite www.unicode.org) e números de ponto flutuante (IEEE 754; para obter mais informações, visite grouper.ieee.org/groups/754/).

C.14 Fundamentos da repetição controlada por contador

Esta seção utiliza a instrução de repetição `while` apresentada na seção C.7 a fim de formalizar os elementos exigidos para realizar a repetição controlada por contador, a qual requer

1. uma **variável de controle** (ou contador do loop)
2. o **valor inicial** da variável de controle
3. o **incremento** (ou **decremento**) pelo qual a variável de controle é modificada sempre que passa pelo loop (conhecido como **cada iteração do loop**)
4. a **condição de continuação do loop** que determina se o loop deve prosseguir.

Para ver esses elementos da repetição controlada por contador, considere o aplicativo da Fig. C.12, o qual utiliza um loop para exibir números de 1 a 10.

```

1 // Fig. C.12: WhileCounter.java
2 // Repetição controlada por contador com a instrução de repetição while.
3
4 public class WhileCounter
5 {
6     public static void main( String[] args )
7     {
8         int counter = 1; // declara e inicializa a variável de controle
9
10        while ( counter <= 10 ) // condição de continuação do loop
11        {
12            System.out.printf( "%d ", counter );
13            ++counter; // incrementa a variável de controle por 1
14        } // fim do while
15
16        System.out.println(); // gera uma nova linha na saída
17    } // fim de main
18 } // fim da classe WhileCounter

```

```
1 2 3 4 5 6 7 8 9 10
```

Figura C.12 Repetição controlada por contador com a instrução de repetição `while`.

Na Fig. C.12, os elementos da repetição controlada por contador são definidos nas linhas 8, 10 e 13. A linha 8 declara a variável de controle (`counter`) como um valor `int`, reserva espaço para ela na memória e configura seu valor inicial como 1. A linha 12 exibe o valor da variável `counter` durante cada iteração do loop. A linha 13 incrementa a variável de controle por 1 a cada iteração do loop. A condição de continuação do loop no `while` (linha 10) testa se o valor da variável de controle é menor ou igual a 10 (o último valor para o qual a condição é verdadeira). O programa executa o corpo desse `while` mesmo quando a variável de controle é 10. O loop termina quando a variável de controle ultrapassa 10 (isto é, `counter` se torna 11).

C.15 Instrução de repetição `for`

A linguagem Java fornece também a **instrução de repetição `for`**, a qual especifica os detalhes da repetição controlada por contador em uma única linha de código. A Figura C.13 reimplementa o aplicativo da Fig. C.12 usando `for`.

```

1 // Fig. C.13: ForCounter.java
2 // Repetição controlada por contador com a instrução de repetição for.
3
4 public class ForCounter
5 {
6     public static void main( String[] args )
7     {
8         // cabeçalho da instrução for inclui inicialização,
9         // condição de continuação do loop e incremento
10        for ( int counter = 1; counter <= 10; ++counter )
11            System.out.printf( "%d ", counter );
12
13        System.out.println(); // gera uma nova linha na saída
14    } // fim de main
15 } // fim da classe For Counter

```

```

1 2 3 4 5 6 7 8 9 10
s

```

Figura C.13 Repetição controlada por contador com a instrução de repetição for.

Quando a instrução for (linhas 10 e 11) começa a ser executada, a variável de controle counter é declarada e inicializada com 1. Em seguida, o programa verifica a condição de continuação do loop, counter <= 10, que está entre os dois pontos e vírgulas exigidos. Como o valor inicial de counter é 1, inicialmente a condição é verdadeira. Portanto, o corpo da instrução (linha 11) exibe o valor da variável de controle counter, ou seja, 1. Depois de executar o corpo do loop, o programa incrementa counter na expressão ++counter, a qual aparece à direita do segundo ponto e vírgula. Então, o teste de continuação do loop é feito novamente para determinar se o programa deve continuar com a próxima iteração. Nesse ponto, o valor da variável de controle é 2, de modo que a condição ainda é verdadeira (o valor final não foi ultrapassado) – portanto, o programa executa novamente o corpo da instrução (isto é, a próxima iteração do loop). Esse processo continua até que os números de 1 a 10 tenham sido exibidos e o valor de counter se torne 11, fazendo o teste de continuação do loop falhar e a repetição terminar (após 10 repetições do corpo do loop). Então, o programa executa a primeira instrução após o for – neste caso, a linha 13.

A Figura C.13 utiliza (na linha 10) a condição de continuação do loop counter <= 10. Se você especificasse incorretamente counter <10 como condição, o loop iteraria apenas nove vezes. Esse erro de lógica comum é chamado de **erro por um**.

Um exame mais detalhado do cabeçalho da instrução for

A Figura C.14 examina com mais detalhes a instrução for da Fig. C.13. A primeira linha do for (incluindo a palavra-chave for e tudo que está entre parênteses depois de for) – linha 10 na Fig. C.13 – às vezes é chamada de **cabeçalho da instrução for**. O cabeçalho de for “faz tudo” – especifica cada item necessário para a repetição controlada por contador com uma variável de controle. Se houver mais de uma instrução no corpo do for, são exigidas chaves para definir o corpo do loop. Se a condição de continuação do loop é inicialmente falsa, o programa não executa o corpo da instrução for – a execução prossegue na instrução após o for.

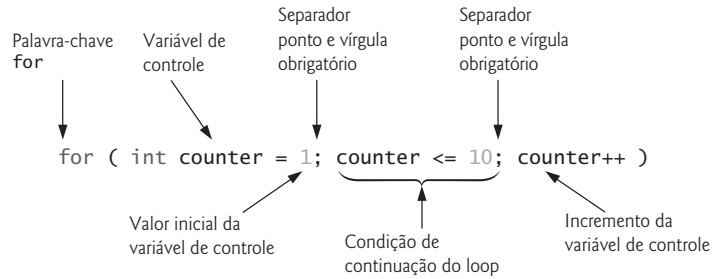


Figura C.14 Componentes do cabeçalho da instrução for.

Escopo da variável de controle de uma instrução for

Se a expressão de *inicialização* no cabeçalho de for declara a variável de controle (isto é, o tipo da variável de controle é especificado antes do nome da variável, como na Fig. C.13), esta *só* pode ser usada nessa instrução for – a variável de controle não vai existir fora dela. Esse uso restrito é conhecido como **escopo** da variável. O escopo de uma variável define onde ela pode ser usada em um programa. Por exemplo, uma variável local *só* pode ser usada no método que a declara e *somente* do ponto da declaração até o final do método.

As expressões no cabeçalho de uma instrução for são opcionais

Todas as três expressões no cabeçalho de um for são opcionais. Se a *condiçãoDeContinuaçãoDoLoop* é omitida, a linguagem Java presume que essa condição é sempre *verdadeira*, criando assim um loop infinito. Você pode omitir a expressão de *inicialização* se o programa inicializa a variável de controle antes do loop. Poderia omitir a expressão de *incremento* se o programa calcula o incremento com instruções no corpo do loop ou se não é necessário um incremento. A expressão de incremento em um for atua como se fosse uma instrução independente no final do corpo do for.

C.16 Exemplos de uso da instrução for

Os exemplos a seguir mostram técnicas para variar a variável de controle em uma instrução for. Em cada caso, escrevemos o cabeçalho de for apropriado. Observe a mudança no operador relacional para loops que *decrementam* a variável de controle a fim de contar para baixo.

- a) Varia a variável de controle de 1 a 100 em *incrementos* de 1.

```
for ( int i = 1; i <= 100; ++i )
```

- b) Varia a variável de controle de 100 a 1 em *decrementos* de 1.

```
for ( int i = 100; i >= 1; --i )
```

- c) Varia a variável de controle de 7 a 77 em *incrementos* de 7.

```
for ( int i = 7; i <= 77; i += 7 )
```

- d) Varia a variável de controle de 20 a 2 em *decrementos* de 2.

```
for ( int i = 20; i >= 2; i -= 2 )
```

- e) Varia a variável de controle pelos valores 2, 5, 8, 11, 14, 17, 20.

```
for ( int i = 2; i <= 20; i += 3 )
```

- f) Varia a variável de controle pelos valores 99, 88, 77, 66, 55, 44, 33, 22, 11, 0.

```
for ( int i = 99; i >= 0; i -= 11 )
```

Aplicação: cálculos de juros compostos

Vamos usar a instrução for para calcular juros compostos. Considere o problema a seguir:

Uma pessoa investe US\$1.000 em uma caderneta de poupança a 5% de juros. Supondo que todo o rendimento seja depositado, calcule e imprima o valor na conta ao final de cada ano, por 10 anos. Use a seguinte fórmula para determinar os valores:

$$a = p (1 + r)^n$$

onde

p é o valor original investido (isto é, o principal)

r é a taxa de juros anual (por exemplo, use 0.05 para 5%)

n é o número de anos

a é o valor depositado ao final do n -ésimo ano.

A solução desse problema (Fig. C.15) envolve um loop que efetua o cálculo indicado para cada um dos 10 anos em que o dinheiro permanece depositado. As linhas 8 a 10 no método main declaram as variáveis double amount, principal e rate, e inicializam principal com 1000.0 e rate com 0.05. A linguagem Java trata constantes de ponto flutuante, como 1000.0 e 0.05, como tipo double. Do mesmo modo, trata constantes inteiras, como 7 e -22, como tipo int.

```
1 // Fig. C.15: Interest.java
2 // Cálculos de juros compostos com for.
3
4 public class Interest
5 {
6     public static void main( String[] args )
7     {
8         double amount; // valor depositado ao final de cada ano
9         double principal = 1000.0; // valor inicial antes dos juros
10        double rate = 0.05; // taxa de juros
11
12        // exibe cabeçalhos
13        System.out.printf( "%s%20s\n", "Year", "Amount on deposit" );
14
15        // calcula o valor depositado para cada um dos dez anos
16        for ( int year = 1; year <= 10; ++year )
17        {
18            // calcula o novo valor para o ano especificado
19            amount = principal * Math.pow( 1.0 + rate, year );
20
21            // exibe o ano e o valor
22            System.out.printf( "%4d%,20.2f\n", year, amount );
23        } // fim do for
24    } // fim de main
25 } // fim da classe Interest
```

Figura C.15 Cálculos de juros compostos com for. (continua)

Year	Amount on deposit
1	1,050.00
2	1,102.50
3	1,157.63
4	1,215.51
5	1,276.28
6	1,340.10
7	1,407.10
8	1,477.46
9	1,551.33
10	1,628.89

Figura C.15 Cálculos de juros compostos com `for`.

Formatando strings com larguras de campo e alinhamento

A linha 13 gera na saída dois cabeçalhos de coluna. A primeira coluna exibe o ano e, a segunda, o valor depositado ao final desse ano. Usamos o especificador de formato `%20s` para gerar a String "Amount on Deposit". O valor 20 inteiro entre `%` e o caractere de conversão `s` indica que o valor deve ser exibido com a **largura de campo** 20 – isto é, `printf` exibe o valor com pelo menos 20 posições de caractere. Se o valor exige menos de 20 posições de caractere (17 neste exemplo), ele é **alinhado à direita** no campo, por padrão. Se o valor de `year` a ser gerado na saída tivesse mais de quatro posições de caractere de largura, a largura do campo seria estendida para a direita a fim de acomodar o valor inteiro – isso empurraria o campo de valor para a direita, desalinhando as colunas de nossa saída tabular. Para produzir valores **alinhados à esquerda** na saída, basta preceder o campo com o **flag de formatação do sinal de subtração** (`-`) como, por exemplo, `%-20s`.

Fazendo os cálculos dos juros

O corpo da instrução `for` (linhas 16 a 23) é executado 10 vezes, variando a variável de controle `year` de 1 a 10 em incrementos de 1. Esse loop termina quando `year` se torna 11. (A variável `year` representa n no enunciado do problema.)

As classes fornecem métodos que realizam tarefas comuns em objetos. Na verdade, a maioria dos métodos deve ser chamada em um objeto específico. Por exemplo, para gerar o texto da Fig. C.15, a linha 13 chama o método `printf` no objeto `System.out`. Muitas classes também fornecem métodos que realizam tarefas comuns e *não* exigem objetos. Eles são chamados de métodos estáticos. Por exemplo, a linguagem Java não contém um operador de exponenciação, de modo que os projetistas da classe Java `Math` definiram o método estático `pow` para elevar um valor a uma potência. Você pode chamar um método estático especificando o nome da classe, seguido por um ponto (`.`) e pelo nome do método, como em

```
NomeDaClasse.nomeDoMétodo( argumentos )
```

No Apêndice D, você vai aprender a implementar métodos estáticos em suas próprias classes.

Usamos o método estático `pow` da classe `Math` para efetuar o cálculo de juros compostos na Fig. C.15. `Math.pow(x, y)` calcula o valor de x elevado à y -ésima potência. O método recebe dois argumentos `double` e retorna um valor `double`. A linha 19 efetua o cálculo $a = p(1 + r)^n$, onde a é `amount`, p é `principal`, r é `rate` e n é `year`. A classe `Math` é definida no pacote `java.lang`, de modo que *não* é necessário importar a classe `Math` para usá-la.

Formatando números de ponto flutuante

Após cada cálculo, a linha 22 gera na saída o ano e o valor depositado ao final desse ano. O ano é gerado com uma largura de campo de quatro caracteres (conforme especificado por %4d). O valor é gerado como um número de ponto flutuante com o especificador de formato %20.2f. O **flag de formatação vírgula (,)** indica que o valor de ponto flutuante deve ser gerado com um **separador de agrupamento**. O separador usado é específico da localidade do usuário (isto é, o país). Por exemplo, nos Estados Unidos, o número será gerado com vírgulas para separar cada três dígitos e um ponto decimal para separar a parte fracionária do número, como em 1,234.45. O número 20 na especificação de formato indica que o valor deve ser gerado na saída alinhado à direita, com uma largura de campo de 20 caracteres. O .2 especifica a precisão do número formatado – neste caso, o número é arredondado para o centésimo mais próximo e gerado na saída com dois dígitos à direita do ponto decimal.

C.17 Instrução de repetição do...while

A **instrução de repetição do...while** é semelhante à instrução while. No while, o programa testa a condição de continuação do loop no início do loop, antes de executar o corpo do loop; se a condição é falsa, o corpo *nunca* executa. A instrução do...while testa a condição de continuação do loop *depois* de executar o corpo do loop; portanto, o *corpo sempre é executado pelo menos uma vez*. Quando uma instrução do...while termina, a execução continua com a próxima instrução na sequência. A Figura C.16 usa um do...while (linhas 10 a 14) para gerar os números de 1 a 10 na saída.

```

1 // Fig. C.16: DoWhileTest.java
2 // Instrução de repetição do...while.
3
4 public class DoWhileTest
5 {
6     public static void main( String[] args )
7     {
8         int counter = 1; // inicializa counter
9
10        do
11        {
12            System.out.printf( "%d ", counter );
13            ++counter;
14        } while ( counter <= 10 ); // fim do do...while
15
16        System.out.println(); // gera uma nova linha na saída
17    } // fim de main
18 } // fim da classe DoWhileTest

```

```

1 2 3 4 5 6 7 8 9 10

```

Figura C.16 Instrução de repetição do...while.

A linha 8 declara e inicializa a variável de controle counter. Ao entrar na instrução do...while, a linha 12 gera o valor de counter e a linha 13 incrementa counter. Então, o programa avalia o teste de continuação do loop no *final* do loop (linha 14). Se a condição é verdadeira, o loop continua a partir da primeira instrução do corpo (linha 12). Se a condição é falsa, o loop termina e o programa continua na próxima instrução após o loop.

C.18 Instrução de seleção múltipla switch

As seções C.5 e C.6 discutiram as instruções de seleção simples `if` e as de seleção dupla `if...else`. A **instrução de seleção múltipla `switch`** executa diferentes ações, de acordo com os valores possíveis de uma **expressão inteira constante** de tipo `byte`, `short`, `int` ou `char`.

Classe `GradeBook` com instruções `switch` para contar conceitos A, B, C, D e F

A Figura C.17 aprimora o estudo de caso de `GradeBook` que começamos a apresentar no Apêndice B. A nova versão que apresentamos agora não apenas calcula a média de um conjunto de notas numéricas digitadas pelo usuário, mas também utiliza uma instrução `switch` para determinar se cada nota é equivalente a um A, B, C, D ou F e para incrementar o contador de notas apropriado. A classe exibe também um resumo do número dos alunos que receberam cada conceito. Consulte a Fig. C.18 para ver exemplos de entradas e saídas do aplicativo `GradeBookTest` que utiliza a classe `GradeBook` para processar um conjunto de notas.

```

1 // Fig. C.17: GradeBook.java
2 // A classe GradeBook usa a instrução switch para contar conceitos em forma de letras.
3 import java.util.Scanner; //
4
5 public class GradeBook
6 {
7     private String courseName; // nome do curso representado por GradeBook
8     // variáveis de instância int são inicializadas com 0 por padrão
9     private int total; // soma das notas
10    private int gradeCounter; // número de notas inseridas
11    private int aCount; // contagem de conceitos A
12    private int bCount; // contagem de conceitos B
13    private int cCount; // contagem de conceitos C
14    private int dCount; // contagem de conceitos D
15    private int fCount; // contagem de conceitos F
16
17    // o construtor inicializa courseName
18    public GradeBook( String name )
19    {
20        courseName = name; // inicializa courseName
21    } // fim do construtor
22
23    // método para configurar o nome do curso
24    public void setCourseName( String name )
25    {
26        courseName = name; // armazena o nome do curso
27    } // fim do método setCourseName
28
29    // método para recuperar o nome do curso
30    public String getCourseName()
31    {
32        return courseName;
33    } // fim do método getCourseName
34
35    // exibe uma mensagem de boas-vindas ao usuário do aplicativo de notas
36    public void displayMessage()
37    {
38        // getCourseName recebe o nome do curso
39        System.out.printf( "Welcome to the grade book for\n%s!\n\n",

```

Figura C.17 A classe `GradeBook` usa a instrução `switch` para contar conceitos em forma de letras.

```

40         getCourseName() );
41     } // fim do método displayMessage
42
43     // insere número arbitrário de notas do usuário
44     public void inputGrades()
45     {
46         Scanner input = new Scanner( System.in );
47
48         int grade; // nota digitada pelo usuário
49
50         System.out.printf( "%s\n%s\n %s\n %s\n",
51             "Enter the integer grades in the range 0-100.",
52             "Type the end-of-file indicator to terminate input:",
53             "On UNIX/Linux/Mac OS X type <Ctrl> d then press Enter",
54             "On Windows type <Ctrl> z then press Enter" );
55
56         // faz loop até o usuário digitar o indicador de fim de arquivo
57         while ( input.hasNext() )
58         {
59             grade = input.nextInt(); // lê grade
60             total += grade; // soma grade ao total
61             ++gradeCounter; // incrementa o número de notas
62
63             // chama método para incrementar o contador apropriado
64             incrementLetterGradeCounter( grade );
65         } // fim do while
66     } // fim do método inputGrades
67
68     // soma 1 ao contador apropriado para a nota especificada
69     private void incrementLetterGradeCounter( int grade )
70     {
71         // determina qual nota foi inserida
72         switch ( grade / 10 )
73         {
74             case 9: // a nota estava entre 90
75             case 10: // e 100, inclusive
76                 ++aCount; // incrementa aCount
77                 break; // necessário para sair do switch
78
79             case 8: // a nota estava entre 80 e 89
80                 ++bCount; // incrementa bCount
81                 break; // sai do switch
82
83             case 7: // a nota estava entre 70 e 79
84                 ++cCount; // incrementa cCount
85                 break; // sai do switch
86
87             case 6: // a nota estava entre 60 e 69
88                 ++dCount; // incrementa dCount
89                 break; // sai do switch
90
91             default: // a nota era menor que 60
92                 ++fCount; // incrementa fCount
93                 break; // opcional; sairá do switch de qualquer forma
94         } // fim de switch
95     } // fim do método incrementLetterGradeCounter
96
97     // exibe um relatório baseado nas notas digitadas pelo usuário
98     public void displayGradeReport()
99     {
100         System.out.println( "\nGrade Report:" );
101     }

```

Figura C.17 A classe GradeBook usa a instrução switch para contar conceitos em forma de letras. (continua)

```

102 // se o usuário digitou pelo menos uma nota...
103 if ( gradeCounter != 0 )
104 {
105     // calcula a média de todas as notas digitadas
106     double average = (double) total / gradeCounter;
107
108     // gera o resumo dos resultados na saída
109     System.out.printf( "Total of the %d grades entered is %d\n",
110         gradeCounter, total );
111     System.out.printf( "Class average is %.2f\n", average );
112     System.out.printf( "%s\n%s%d\n%s%d\n%s%d\n%s%d\n%s%d\n",
113         "Number of students who received each grade:",
114         "A: ", aCount, // exibe o número de conceitos A
115         "B: ", bCount, // exibe o número de conceitos B
116         "C: ", cCount, // exibe o número de conceitos C
117         "D: ", dCount, // exibe o número de conceitos D
118         "F: ", fCount ); // exibe o número de conceitos F
119 } // fim do if
120 else // nenhuma nota foi inserida; portanto, gera a mensagem apropriada na saída
121     System.out.println( "No grades were entered" );
122 } // fim do método displayGradeReport
123 } // fim da classe GradeBook

```

Figura C.17 A classe GradeBook usa a instrução switch para contar conceitos em forma de letras.

Assim como as versões anteriores, a classe GradeBook (Fig. C.17) declara a variável de instância `courseName` (linha 7) e contém os métodos `setCourseName` (linhas 24 a 27), `getCourseName` (linhas 30 a 33) e `displayMessage` (linhas 36 a 41), os quais respectivamente configuram o nome do curso, armazenam esse nome e exibem uma mensagem de boas-vindas para o usuário. A classe contém também um construtor (linhas 18 a 21) que inicializa o nome do curso.

A classe GradeBook declara ainda as variáveis de instância `total` (linha 9) e `gradeCounter` (linha 10), as quais monitoram a soma das notas digitadas pelo usuário e o número de notas inseridas, respectivamente. As linhas 11 a 15 declaram variáveis contadoras para cada categoria de nota. A classe GradeBook mantém `total`, `gradeCounter` e os cinco contadores de conceito como variáveis de instância, para que possam ser usadas ou modificadas em qualquer um dos métodos da classe. O construtor da classe (linhas 18 a 21) configura somente o nome do curso, pois as sete variáveis de instância restantes são valores `int` e inicializadas com 0 por padrão.

A classe GradeBook contém três métodos adicionais – `inputGrades`, `incrementLetterGradeCounter` e `displayGradeReport`. O método `inputGrades` (linhas 44 a 66) lê um número arbitrário de notas inteiras do usuário, utilizando repetição controlada por sentinela, e atualiza as variáveis de instância `total` e `gradeCounter`. Esse método chama o método `incrementLetterGradeCounter` (linhas 69 a 95) para atualizar o contador de conceito apropriado para cada nota inserida. O método `displayGradeReport` (linhas 98 a 122) gera na saída um relatório contendo o total de todas as notas inseridas, a média das notas e o número de alunos que receberam cada conceito. Vamos examinar esses métodos com mais detalhes.

Método `inputGrades`

A linha 48 no método `inputGrades` declara a variável `grade` que vai armazenar a entrada do usuário. As linhas 50 a 54 pedem ao usuário para que insira notas inteiras e digite o indicador de fim de arquivo para finalizar a entrada. O **indicador de fim de arquivo** é uma combinação de toques de tecla dependente do sistema, que o usuário insere para

indicar que não há mais dados a digitar. Em sistemas UNIX/Linux/Mac OS X, o fim de arquivo é inserido digitando-se a sequência

```
<Ctrl> d
```

sozinha em uma linha. Essa notação significa pressionar a tecla *Ctrl* e a tecla *d* simultaneamente. Em sistemas Windows, o fim de arquivo pode ser inserido digitando-se

```
<Ctrl> z
```

[*Obs.:* em alguns sistemas, você deve pressionar *Enter* depois de digitar a sequência de teclas de fim de arquivo. Além disso, o Windows normalmente exibe os caracteres ^Z na tela, quando o indicador de fim de arquivo é digitado, como mostrado na saída da Fig. C.18.]

A instrução `while` (linhas 57 a 65) obtém a entrada do usuário. A condição na linha 57 chama o método `hasNext` de `Scanner` para determinar se há mais dados para inserir. Esse método retorna o valor booleano `true` se há mais dados; caso contrário, retorna `false`. Então, o valor retornado é usado como valor da condição na instrução `while`. O método `hasNext` retorna `false` quando o usuário digita o indicador de fim de arquivo.

A linha 59 insere um valor de nota do usuário. A linha 60 soma `grade` a `total`. A linha 61 incrementa `gradeCounter`. O método `displayGradeReport` da classe utiliza essas variáveis para calcular a média das notas. A linha 64 chama o método `incrementLetterGradeCounter` da classe (declarado nas linhas 69 a 95) para incrementar o contador de conceitos apropriado com base na nota numérica digitada.

Método `incrementLetterGradeCounter`

O método `incrementLetterGradeCounter` contém uma instrução `switch` (linhas 72 a 94) que determina qual contador vai ser incrementado. Supomos que o usuário digita uma nota válida no intervalo de 0 a 100. Uma nota no intervalo de 90 a 100 representa A, de 80 a 89 representa B, de 70 a 79 representa C, de 60 a 69 representa D e de 0 a 59 representa F. A instrução `switch` consiste em um bloco que contém uma sequência de **rótulos `case`** e um **`case default`** opcional. Eles são usados neste exemplo para determinar qual contador vai ser incrementado com base na nota.

Quando o fluxo de controle chega ao `switch`, o programa avalia a expressão nos parênteses (`grade / 10`) após a palavra-chave `switch`. Essa é a **expressão de controle** do `switch`. O programa compara o valor dessa expressão (que deve ser avaliada com um valor inteiro de tipo `byte`, `char`, `short` ou `int`) com cada rótulo `case`. A expressão de controle na linha 72 efetua a divisão inteira, a qual *trunca a parte fracionária* do resultado. Assim, quando dividimos um valor de 0 a 100 por 10, o resultado é sempre um valor de 0 a 10. Usamos vários desses valores em nossos rótulos `case`. Por exemplo, se o usuário digita o valor inteiro 85, a expressão de controle é avaliada como 8. O `switch` compara 8 com cada rótulo `case`. Se ocorre uma correspondência (`case 8:` na linha 79), o programa executa as instruções desse `case`. Para o valor inteiro 8, a linha 80 incrementa `bCount`, pois uma nota na faixa dos 80 é um B. A **instrução `break`** (linha 81) faz o controle do programa passar para a primeira instrução após o `switch` – neste programa, atingimos o fim do corpo do método `incrementLetterGradeCounter`; portanto, o método termina e o controle retorna para a linha 65 no método `inputGrades` (a primeira linha após a chamada de `incrementLetterGradeCounter`). A linha 65 é o fim do corpo de um loop `while`; portanto, o controle flui para a condição do `while` (linha 57) para determinar se o loop deve continuar a ser executado.

Os cases em nosso switch testam explicitamente os valores 10, 9, 8, 7 e 6. Observe os cases nas linhas 74 a 75, que testam os valores 9 e 10 (ambos os quais representam o conceito A). Listar cases consecutivamente dessa maneira, sem uma instrução entre eles, permite que os cases executem o mesmo conjunto de instruções – quando a expressão de controle for avaliada como 9 ou 10, as instruções das linhas 76 e 77 serão executadas. A instrução switch não oferece um mecanismo para testar intervalos de valores; portanto, todo valor que você precisar testar deverá ser listado em um rótulo case separado. Cada case pode ter várias instruções. A instrução switch é diferente das outras instruções de controle, pois *não* exige chaves em torno das várias instruções em um case.

Sem instruções break, sempre que ocorre uma correspondência no switch, as instruções desse case e dos cases subsequentes são executadas até que seja encontrada uma instrução break ou o final do switch. (Essa característica é útil para escrever um programa conciso que exiba a música repetitiva “The Twelve Days of Christmas”).

Se não ocorre correspondência entre o valor da expressão de controle e um rótulo case, o case default (linhas 91 a 93) é executado. Usamos o case default neste exemplo para processar todos os valores da expressão de controle menores que 6 – isto é, todas as notas de reprovação. Se não ocorre correspondência e o switch não contém um case default, o controle do programa simplesmente continua na primeira instrução após o switch.

Classe GradeBookTest que demonstra a classe GradeBook

A classe GradeBookTest (Fig. C.18) cria um objeto GradeBook (linhas 10 e 11). A linha 13 chama o método showMessage do objeto para mostrar na saída uma mensagem de boas-vindas para o usuário. A linha 14 chama o método inputGrades do objeto para ler um conjunto de notas do usuário e monitorar a soma de todas as notas inseridas e o número de notas. Lembre-se de que o método inputGrades chama também o método incrementLetterGradeCounter para monitorar o número dos alunos que receberam cada conceito. A linha 15 chama o método displayGradeReport da classe GradeBook, o qual gera na saída um relatório baseado nas notas inseridas (como na janela de entrada/saída da Fig. C.18). A linha 103 da classe GradeBook (Fig. C.17) determina se o usuário digitou pelo menos uma nota – isso nos ajuda a evitar uma divisão por zero. Em caso positivo, a linha 106 calcula a média das notas. Então, as linhas 109 a 118 geram na saída o total de todas as notas, a média da classe e o número de alunos que receberam cada conceito. Se nenhuma nota foi inserida, a linha 121 gera uma mensagem apropriada na saída. A saída na Fig. C.18 mostra um exemplo de relatório de notas baseado em 10 notas.

```

1 // Fig. C.18: GradeBookTest.java
2 // Cria objeto GradeBook, insere as notas e exibe o relatório de notas.
3
4 public class GradeBookTest
5 {
6     public static void main( String[] args )
7     {
8         // cria objeto myGradeBook de GradeBook e
9         // passa o nome do curso para o construtor
10        GradeBook myGradeBook = new GradeBook(
11            "CS101 Introduction to Java Programming" );
12
13        myGradeBook.showMessage(); // exibe a mensagem de boas-vindas

```

Figura C.18 Cria objeto GradeBook, insere as notas e exibe o relatório de notas.

```

14     myGradeBook.inputGrades(); // lê as notas do usuário
15     myGradeBook.displayGradeReport(); // exibe relatório baseado nas notas
16 } // fim de main
17 } // fim da classe GradeBookTest

```

```

Welcome to the grade book for
CS101 Introduction to Java Programming!

Enter the integer grades in the range 0-100.
Type the end-of-file indicator to terminate input:
  On UNIX/Linux/Mac OS X type <Ctrl> d then press Enter
  On Windows type <Ctrl> z then press Enter
99
92
45
57
63
71
76
85
90
100
^Z

Grade Report:
Total of the 10 grades entered is 778
Class average is 77.80

Number of students who received each grade:
A: 4
B: 1
C: 2
D: 1
F: 2

```

Figura C.18 Cria objeto GradeBook, insere as notas e exibe o relatório de notas.

A classe GradeBookTest (Fig. C.18) não chama o método `incrementLetterGradeCounter` de GradeBook diretamente (linhas 69 a 95 da Fig. C.17). Esse método é usado exclusivamente pelo método `inputGrades` da classe GradeBook para atualizar o contador de conceito apropriado à medida que cada nova nota é inserida pelo usuário. O método `incrementLetterGradeCounter` existe unicamente para oferecer suporte para as operações de outros métodos de GradeBook; portanto, é declarado `private`.

A instrução `break` não é exigida para o último case do `switch` (ou para o case `default` opcional, quando aparece por último), pois a execução continua na próxima instrução após o `switch`.

Observações sobre a expressão em cada case de uma instrução switch

Quando usar a instrução `switch`, lembre-se de que cada case deve conter uma expressão inteira constante – isto é, qualquer combinação de constantes inteiras que seja avaliada como um valor inteiro constante (por exemplo, `-7`, `0` ou `221`). Uma constante inteira é simplesmente um valor inteiro. Além disso, você pode usar **constantes de caractere** – caracteres específicos entre aspas simples, como `'A'`, `'7'` ou `'$'` –, as quais representam os valores inteiros de caracteres e constantes `enum` (apresentadas na seção D.10).

Em cada case, a expressão também pode ser uma **variável constante** — uma variável contendo um valor que não muda no programa inteiro. Tal variável é declarada com a palavra-chave `final` (discutida no Apêndice D). A linguagem Java tem um recurso chamado *enumeração*, o qual também apresentamos no Apêndice D. As constantes de enumeração também podem ser usadas em rótulos case.

Usando objetos *String* em instruções *switch* (novidade do Java SE 7)

A partir do Java SE 7, é possível usar objetos *String* na expressão de controle e em rótulos case de uma instrução *switch*. Por exemplo, talvez você queira usar o nome de uma cidade para obter o código postal correspondente. Supondo que *city* e *zipCode* sejam variáveis *String*, a seguinte instrução *switch* executa essa tarefa para três cidades:

```
switch( city )
{
    case "Maynard":
        zipCode = "01754";
        break;
    case "Marlborough":
        zipCode = "01752";
        break;
    case "Framingham":
        zipCode = "01701";
        break;
} // fim do switch
```

C.19 Instruções *break* e *continue*

Além das instruções de seleção e repetição, a linguagem Java fornece as instruções *break* e *continue* para alterar o fluxo de controle. A seção anterior mostrou como *break* pode ser usada para terminar a execução de uma instrução *switch*. Esta seção discute como usar *break* em instruções de repetição.

Instrução *break*

A instrução *break*, quando executada em um *while*, *for*, *do...while* ou *switch*, causa a saída imediata dessa instrução. A execução continua na primeira instrução após a instrução de controle. A instrução *break* é comumente usada para sair prematuramente de um loop ou para pular o restante de um *switch*.

Instrução *continue*

A instrução *continue*, quando executada em um *while*, *for* ou *do...while*, pula as instruções restantes no corpo do loop e prossegue na *próxima iteração* do loop. Em instruções *while* e *do...while*, o programa avalia o teste de continuação do loop imediatamente após a execução da instrução *continue*. Em uma instrução *for*, a expressão de incremento é executada e, então, o programa avalia o teste de continuação do loop.

C.20 Operadores lógicos

Os **operadores lógicos** da linguagem Java permitem formar expressões mais complexas pela *combinação* de condições simples. São eles: `&&` (E condicional), `||` (OU condicional), `&` (E lógico booleano), `|` (OU inclusivo lógico booleano), `^` (OU exclusivo lógico booleano) e `!` (NÃO lógico). [*Obs.: &, | e ^ também são operadores bit a bit, quando aplicados a operandos inteiros.*]

Operador E condicional (&&)

Suponha que, em algum ponto de um programa, queiramos que duas condições sejam *ambas* verdadeiras, antes de escolhermos determinado caminho de execução. Nesse caso, podemos usar o operador **&& (E condicional)**, como segue:

```
if ( gender == FEMALE && age >= 65 )
    ++seniorFemales;
```

Essa instrução `if` contém duas condições simples. A condição `gender == FEMALE` compara a variável `gender` com a constante `FEMALE` para determinar se uma pessoa é mulher (female). A condição `age >= 65` poderia ser avaliada para determinar se uma pessoa é idosa. A instrução `if` considera a condição combinada

```
gender == FEMALE && age >= 65
```

que é verdadeira se e somente se *as duas* condições simples são verdadeiras. Nesse caso, o corpo da instrução `if` incrementa `seniorFemales` por 1. Se uma das condições simples ou ambas forem falsas, o programa pula o incremento. Alguns programadores acham a condição combinada anterior mais legível quando são adicionados parênteses redundantes, como em:

```
( gender == FEMALE ) && ( age >= 65 )
```

A tabela da Fig. C.19 resume o operador **&&**. Ela mostra quatro combinações possíveis de valores `false` e `true` para *expressão1* e *expressão2*. Tais tabelas são chamadas de **tabelas-verdade**. A linguagem Java avalia como `false` ou como `true` todas as expressões que incluem operadores relacionais, operadores de igualdade ou operadores lógicos.

expressão1	expressão2	expressão1 && expressão2
false	false	false
false	true	false
true	false	false
true	true	true

Figura C.19 Tabela-verdade do operador **&&** (E condicional).

Operador OU condicional (||)

Agora, suponha que queiramos garantir que duas condições sejam *uma delas* ou *ambas* verdadeiras, antes de escolhermos determinado caminho de execução. Nesse caso, usamos o operador **|| (OU condicional)**, como no trecho de programa a seguir:

```
if ( ( semesterAverage >= 90 ) || ( finalExam >= 90 ) )
    System.out.println ( "Student grade is A" );
```

Essa instrução também contém duas condições simples. A condição `semesterAverage >= 90` é avaliada para determinar se o aluno merece A no curso devido a um bom desempenho ao longo do semestre. A condição `finalExam >= 90` é avaliada para determinar se o aluno merece A no curso devido a um desempenho excelente no exame final. Então, a instrução `if` considera a condição combinada

```
( semesterAverage >= 90 ) || ( finalExam >= 90 )
```

e premia o aluno com um A se *uma ou ambas* as condições simples são verdadeiras. A única vez em que a mensagem "Student grade is A" *não* é impressa é quando *as duas* condições simples são *falsas*. A Figura C.20 é a tabela-verdade do operador OU condicional (||). O operador && tem precedência mais alta que o operador ||. Os dois operadores são associados da esquerda para a direita.

expressão1	expressão2	expressão1 expressão2
false	false	false
false	true	true
true	false	true
true	true	true

Figura C.20 Tabela-verdade do operador || (OU condicional).

Avaliação de curto-circuito de condições complexas

As partes de uma expressão contendo operadores && ou || são avaliadas *somente* até se saber se a condição é verdadeira ou falsa. Assim, a avaliação da expressão

```
( gender == FEMALE ) && ( age >= 65 )
```

para imediatamente se gender não é igual a FEMALE (isto é, toda a expressão é false) e continua se gender é igual a FEMALE (isto é, toda a expressão ainda pode ser true se a condição age >= 65 for true). Essa característica das expressões E condicionais e OU condicionais é chamada de **avaliação de curto-circuito**.

Operadores E lógico booleano (&) e OU inclusivo lógico booleano (|)

Os operadores **E lógico booleano (&)** e **OU inclusivo lógico booleano (|)** são idênticos aos operadores & e ||, exceto que os operadores & e | avaliam *sempre* seus *dois* operandos (isto é, eles *não* fazem avaliação de curto-circuito). Assim, a expressão

```
( gender == 1 ) & ( age >= 65 )
```

avalia age >= 65 *independentemente* de gender ser igual a 1. Isso é útil quando o operando da direita do operador E lógico booleano ou OU inclusivo lógico booleano tem um **efeito colateral** exigido – uma modificação no valor de uma variável. Por exemplo, a expressão

```
( birthday == true ) | ( ++age >= 65 )
```

garante que a condição ++age >= 65 será avaliada. Assim, a variável age é incrementada, independentemente de a expressão global ser true ou false.



Dica de prevenção de erro C.1

Por clareza, evite expressões com efeitos colaterais nas condições. Os efeitos colaterais parecem engenhosos, mas podem dificultar o entendimento do código e levar a erros de lógica sutis.

OU exclusivo lógico booleano (^)

Uma condição simples contendo o operador **OU exclusivo lógico booleano (^)** é true se e somente se um de seus operandos for true e o outro false. Se ambos são true ou ambos são false, a condição inteira é false. A Figura C.21 é a tabela-verdade do operador OU exclusivo (^) booleano. Esse operador avalia sempre os seus *dois* operandos.

expressão1	expressão2	expressão1 ^ expressão2
false	false	false
false	true	true
true	false	true
true	true	false

Figura C.21 Tabela-verdade do operador ^ (OU exclusivo lógico booleano).

Operador de negação lógico (!)

O operador ! (**NÃO lógico**, também chamado de **negação lógica** ou **complemento lógico**) “inverte” o significado de uma condição. Ao contrário dos operadores lógicos &&, ||, &, | e ^, que são *binários* e combinam duas condições, o operador de negação lógico é *unário* e tem apenas uma condição como operando. Ele é colocado *antes* de uma condição para escolher um caminho de execução, caso a condição original (sem o operador de negação lógico) seja false, como neste trecho de programa

```
if ( ! ( grade == sentinelValue ) )
    System.out.printf( "The next grade is %d\n", grade );
```

que só executa a chamada de printf se grade *não* for igual a sentinelValue. Os parênteses em torno da condição grade == sentinelValue são necessários porque o operador de negação lógico tem precedência mais alta que o operador de igualdade.

Na maioria dos casos, é possível evitar o uso da negação lógica expressando a condição de um modo diferente, com um operador relacional ou de igualdade apropriado. Por exemplo, a instrução anterior também pode ser escrita como segue:

```
if ( grade != sentinelValue )
    System.out.printf( "The next grade is %d\n", grade );
```

Essa flexibilidade pode ajudá-lo a expressar uma condição de uma maneira mais conveniente. A Figura C.22 é a tabela-verdade do operador de negação lógico.

expressão	!expressão
false	true
true	false

Figura C.22 Tabela-verdade do operador ! (negação lógica ou NÃO lógico).

C.21 Para finalizar

Este apêndice apresentou a solução de problemas básicos na construção de classes e no desenvolvimento de métodos para essas classes. Demonstramos como construir um algoritmo (isto é, uma estratégia para resolver um problema) e, depois, como refiná-lo por meio de várias fases de desenvolvimento com pseudocódigo, resultando em código Java que pode ser executado como parte de um método. O apêndice mostrou como usar refinamento gradual, de cima para baixo, para planejar as ações específicas a serem executadas por um método e a ordem na qual o método deve executá-las.

Apenas três tipos de estruturas de controle – sequência, seleção e repetição – são necessárias para desenvolver qualquer algoritmo de solução de problema. Especificamente, este apêndice demonstrou a instrução de seleção simples if, a instrução de se-

leção dupla `if...else` e a instrução de repetição `while`. Esses são alguns dos elementos fundamentais utilizados na construção de soluções para muitos problemas. Utilizamos empilhamento de instruções de controle para totalizar e tirar a média de um conjunto de notas de alunos com repetição controlada por contador e por sentinela, e usamos aninhamento de instruções de controle para analisar e tomar decisões baseadas em um conjunto de resultados de exame. Apresentamos os operadores de atribuição compostos do Java e seus operadores de incremento e decremento. Discutimos os tipos primitivos da linguagem Java.

Demonstramos as instruções `for`, `do...while` e `switch`. Mostramos que qualquer algoritmo pode ser desenvolvido com combinações da estrutura de sequência (isto é, instruções listadas na ordem em que devem executar), com os três tipos de instruções de seleção – `if`, `if...else` e `switch` – e com os três tipos de instruções de repetição – `while`, `do...while` e `for`. Discutimos como combinar esses elementos fundamentais para utilizar técnicas comprovadas de construção de programas e solução de problemas. Apresentamos também os operadores lógicos da linguagem Java, os quais permitem usar expressões condicionais mais complexas em instruções de controle. No Apêndice D, examinaremos os métodos com mais profundidade.

Exercícios de revisão (seções C.1 a C.13)

C.1 Preencha os espaços em branco em cada um dos seguintes enunciados:

- Todos os programas podem ser escritos em termos de três tipos de estruturas de controle: _____, _____ e _____.
- A instrução _____ é usada para executar uma ação quando uma condição é verdadeira e outra ação quando essa condição é falsa.
- Quando não se sabe antecipadamente quantas vezes um conjunto de instruções será repetido, um valor de _____ pode ser usado para terminar a repetição.
- Java é uma linguagem _____; ela exige que todas as variáveis tenham um tipo.
- Se o operador de incremento é _____ a uma variável, primeiramente a variável é incrementada por 1 e depois seu novo valor é usado na expressão.

C.2 Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.

- Um conjunto de instruções contidas dentro de um par de parênteses é chamado de bloco.
- Uma instrução de seleção especifica que uma ação deve ser repetida enquanto alguma condição permanecer verdadeira.
- Uma instrução de controle aninhada aparece no corpo de outra instrução de controle.
- Especificar a ordem na qual as instruções são executadas em um programa é chamado de controle do programa.
- As variáveis de instância de tipo `boolean` recebem o valor `true` por padrão.

C.3 Escreva instruções em Java que realizem cada uma das tarefas a seguir:

- Use uma instrução para atribuir a soma de `x` e `y` a `z` e, então, incremente `x` por 1.
- Teste se a variável `count` é maior que 10. Se for, imprima "Count is greater than 10".
- Use uma instrução para decrementar a variável `x` por 1 e, então, a subtraia da variável `total` e armazene o resultado na variável `total`.
- Calcule o resto após `q` ser dividido por `divisor` e atribua o resultado a `q`. Escreva essa instrução de duas maneiras diferentes.

- C.4** Escreva uma instrução em Java para realizar cada uma das tarefas a seguir:
- Declarar as variáveis `sum` e `x` como sendo de tipo `int`.
 - Atribuir 1 à variável `x`.
 - Atribuir 0 à variável `sum`.
 - Somar a variável `x` à variável `sum` e atribuir o resultado à variável `sum`.
 - Imprimir "The sum is: ", seguido do valor da variável `sum`.
- C.5** Determine o valor das variáveis na instrução `product *= x++`; após o cálculo ser efetuado. Presuma que todas as variáveis são de tipo `int` e inicialmente têm o valor 5.
- C.6** Identifique e corrija os erros em cada um dos seguintes conjuntos de código:
- ```
while (c <= 5)
{
 product *= c;
 ++c;
}
```
  - ```
if ( gender == 1 )
    System.out.println( "Woman" );
else;
    System.out.println( "Man" );
```
- C.7** O que está errado na instrução `while` a seguir?
- ```
while (z >= 0)
 sum += z;
```

## Exercícios de revisão (seções C.14 a C.20)

- C.8** Preencha os espaços em branco em cada um dos seguintes enunciados:
- Normalmente, instruções \_\_\_\_\_ são usadas para repetição controlada por contador e instruções \_\_\_\_\_ para repetição controlada por sentinela.
  - A instrução `do...while` testa condição de continuação do loop \_\_\_\_\_ executar o corpo do loop; portanto, o corpo sempre é executado pelo menos uma vez.
  - A instrução \_\_\_\_\_ faz uma seleção dentre várias ações com base nos valores possíveis de uma variável ou expressão inteira.
  - O operador \_\_\_\_\_ pode ser usado para garantir que duas condições sejam *ambas* verdadeiras, antes de escolher determinado caminho de execução.
  - Se a condição de continuação do loop em um cabeçalho `for` é inicialmente \_\_\_\_\_, o programa não executa o corpo da instrução `for`.
- C.9** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- O `case default` é exigido na instrução de seleção `switch`.
  - A instrução `break` é exigida no último `case` de uma instrução de seleção `switch`.
  - A expressão `( x > y ) && ( a < b )` é verdadeira se `x > y` é verdadeira ou se `a < b` é verdadeira.
  - Uma expressão contendo o operador `||` é verdadeira se um de seus operandos ou ambos são verdadeiros.
  - Listar `cases` consecutivamente, sem uma instrução entre eles, permite que os `cases` executem o mesmo conjunto de instruções.
- C.10** Escreva uma instrução ou um conjunto de instruções em Java para realizar cada uma das tarefas a seguir:
- Somar os inteiros ímpares entre 1 e 99 usando uma instrução `for`. Suponha que as variáveis inteiras `sum` e `count` tenham sido declaradas.
  - Calcular o valor de 2.5 elevado à potência de 3 usando o método `pow`.

- c) Imprimir os inteiros de 1 a 20 usando um loop `while` e a variável contadora `i`. Suponha que a variável `i` tenha sido declarada, mas não inicializada. Imprima apenas cinco inteiros por linha. [*Dica:* use o cálculo `i % 5`. Quando o valor dessa expressão for 0, imprima um caractere de nova linha; caso contrário, imprima um caractere de tabulação. Suponha que esse código seja um aplicativo. Use o método `System.out.println()` para gerar o caractere de nova linha e o método `System.out.print( '\t' )` para gerar o caractere de tabulação.]
- d) Repita a parte (c) usando uma instrução `for`.

**C.11** Encontre o erro em cada um dos trechos de código a seguir e explique como corrigi-lo:

- a) 

```
i = 1;
while (i <= 10);
 ++i;
}
```
- b) 

```
for (k = 0.1; k != 1.0; k += 0.1)
 System.out.println(k);
```
- c) 

```
switch (n)
{
 case 1:
 System.out.println("The number is 1");
 case 2:
 System.out.println("The number is 2");
 break;
 default:
 System.out.println("The number is not 1 or 2");
 break;
}
```
- d) O código a seguir deve imprimir os valores de 1 a 10:  

```
n = 1;
while (n < 10)
 System.out.println(n++);
```

## Respostas dos exercícios de revisão (seções C.1 a C.13)

- C.1** a) sequência, seleção, repetição. b) `if...else`. c) sentinela, sinal, flag ou dummy. d) fortemente tipada. e) prefixado.
- C.2** a) Falsa. Um conjunto de instruções contidas dentro de um par de chaves (`{}`) é chamado de bloco. b) Falsa. Uma instrução de repetição específica que uma ação deve ser repetida enquanto alguma condição permanecer verdadeira. c) Verdadeira. d) Verdadeira. e) Falsa. As variáveis de instância de tipo `boolean` recebem o valor `false` por padrão.
- C.3** a) `z = x++ + y;`  
b) 

```
if (count > 10)
 System.out.println("Count is greater than 10");
```

  
c) `total -= --x;`  
d) `q %= divisor;`  
`q = q % divisor;`
- C.4** a) 

```
int sum;
int x;
```

  
b) `x = 1;`  
c) `sum = 0;`  
d) `sum += x; OR sum = sum + x;`  
e) `System.out.printf( "The sum is: %d\n", sum );`

- C.5** `product = 25, x = 6`
- C.6** a) Erro: está faltando a chave de fechamento do corpo da instrução `while`.  
Correção: acrescentar uma chave de fechamento depois da instrução `++c;`.  
b) Erro: o ponto e vírgula após o `else` resulta em um erro de lógica. A segunda instrução de saída sempre será executada.  
Correção: remover o ponto e vírgula após o `else`.
- C.7** O valor da variável `z` nunca é alterado na instrução `while`. Portanto, se a condição de continuação do loop (`z >= 0`) for verdadeira, será criado um loop infinito. Para evitar que ocorra um loop infinito, `z` deve ser decrementada a fim de que finalmente se torne menor que 0.

## Respostas dos exercícios de revisão (seções C.14 a C.20)

- C.8** a) `for`, `while`. b) após. c) `switch`. d) `continue`. e) falsa.
- C.9** a) Falsa. O `case default` é opcional. Se não for necessária uma ação padrão, não há necessidade de um `case default`. b) Falsa. A instrução `break` é usada para sair da instrução `switch`. A instrução `break` não é exigida no último `case` de uma instrução `switch`. c) Falsa. Ao se usar o operador `&&`, as duas expressões relacionais devem ser verdadeiras para que toda a expressão seja verdadeira. d) Verdadeira. e) Verdadeira.
- C.10** a) `sum = 0;`  
`for ( count = 1; count <= 99; count += 2 )`  
`sum += count;`
- b) `double result = Math.pow( 2.5, 3 );`
- c) `i = 1;`
- ```
while ( i <= 20 )
{
    System.out.print( i );

    if ( i % 5 == 0 )
        System.out.println();
    else
        System.out.print( '\t' );

    ++i;
}
```
- d) `for ( i = 1; i <= 20; ++i )`
- ```
{
 System.out.print(i);

 if (i % 5 == 0)
 System.out.println();
 else
 System.out.print('\t');
}
```
- C.11** a) Erro: o ponto e vírgula após o cabeçalho de `while` causa um loop infinito e falta uma chave de fechamento.  
Correção: substituir o ponto e vírgula por `{` ou remover o `;` e o `}`.



- b) Erro: usar um número de ponto flutuante para controlar uma instrução `for` pode não funcionar, pois os números de ponto flutuante são representados apenas aproximadamente pela maioria dos computadores.

Correção: usar um valor inteiro e efetuar o cálculo correto para obter os valores desejados:

```
for (k = 1; k != 10; ++k)
 System.out.println((double) k / 10);
```

- c) Erro: o código ausente é a instrução `break` nas instruções do primeiro `case`. Correção: adicionar uma instrução `break` ao final das instruções do primeiro `case`. Essa omissão não é necessariamente um erro se você quer que a instrução de `case 2`: execute sempre que a instrução de `case 1`: for executada.
- d) Erro: uso do operador relacional incorreto na condição de continuação do `while`. Correção: usar `<=` em vez de `<` ou alterar 10 para 11.

## Exercícios (seções C.1 a C.13)

**C.12** Explique o que acontece quando um programa em Java tenta dividir um valor inteiro por outro. O que acontece com a parte fracionária do cálculo? Como você pode evitar esse efeito?

**C.13** Descreva as duas maneiras pelas quais as instruções de controle podem ser combinadas.

**C.14** Que tipo de repetição seria adequada para calcular a soma dos primeiros 100 inteiros positivos? Que tipo seria adequado para calcular a soma de um número arbitrário de inteiros positivos? Descreva sucintamente como cada uma dessas tarefas poderia ser realizada.

**C.15** Qual é a diferença entre pré-incrementar e pós-incrementar uma variável?

**C.16** Identifique e corrija os erros em cada um dos seguintes trechos de código: [*Obs.:* pode existir mais de um erro em cada trecho.]

- a) 

```
if (age >= 65);
 System.out.println("Age is greater than or equal to 65");
else
 System.out.println("Age is less than 65");
```
- b) 

```
int x = 1, total;
while (x <= 10)
{
 total += x;
 ++x;
}
```
- c) 

```
while (x <= 100)
 total += x;
 ++x;
```
- d) 

```
while (y > 0)
{
 System.out.println(y);
 ++y;
```

Para os exercícios C.17 e C.18, execute cada um dos passos a seguir:

- Leia o enunciado do problema.
- Escreva um programa em Java.
- Teste, depure e execute o programa.
- Processe três conjuntos de dados completos.

**C.17 (Consumo de combustível)** Os motoristas se preocupam com o consumo de seus carros. Um motorista monitorou várias viagens, registrando os quilômetros percorridos e os litros de combustível utilizados de cada tanque cheio. Desenvolva um aplicativo em Java que insira os quilômetros percorridos e os litros de combustível utilizados (ambos como valores inteiros) em cada viagem. O programa deve calcular e exibir a quilometragem por litro obtida em cada viagem e imprimir os quilômetros por litro combinados de todas as viagens até esse ponto. Todos os cálculos de média devem produzir resultados de ponto flutuante. Use a classe `Scanner` e repetição controlada por sentinela para obter os dados do usuário.

**C.18 (Calculadora de limite de crédito)** Desenvolva um aplicativo em Java que determine se algum dos vários clientes de uma loja de departamentos ultrapassou seu limite de crédito. Para cada cliente estão disponíveis os seguintes dados:

- número da conta
- saldo no início do mês
- total de todos os itens cobrados pelo cliente nesse mês
- total de todos os créditos aplicados à conta do cliente nesse mês
- limite de crédito permitido

O programa deve inserir todos esses dados como valores inteiros, calcular o novo saldo (= *saldo inicial + cobranças – créditos*), exibir o novo saldo e determinar se ele ultrapassa o limite de crédito do cliente. Para os clientes cujos limites foram ultrapassados, o programa deve exibir a mensagem "Credit limit exceeded".

**C.19 (Encontrar o maior número)** O processo de encontrar o maior valor é usado frequentemente em aplicativos de computador. Por exemplo, um programa que determina o vencedor de uma competição de vendas inseriria o número de unidades comercializadas por cada vendedor. O vendedor que vende mais unidades ganha o concurso. Escreva um programa em pseudocódigo e depois um aplicativo Java que insira uma série de 10 valores inteiros, determine e imprima o maior valor. Seu programa deve usar pelo menos as três variáveis a seguir:

- counter: um contador para contar até 10 (isto é, para monitorar quantos números foram inseridos e determinar quando todos os 10 foram processados).
- number: o valor inteiro inserido mais recentemente pelo usuário.
- largest: o maior número encontrado até o momento.

**C.20 (Saída tabular)** Escreva um aplicativo Java que utilize um loop para imprimir a seguinte tabela de valores:

| N | 10*N | 100*N | 1000*N |
|---|------|-------|--------|
| 1 | 10   | 100   | 1000   |
| 2 | 20   | 200   | 2000   |
| 3 | 30   | 300   | 3000   |
| 4 | 40   | 400   | 4000   |
| 5 | 50   | 500   | 5000   |

**C.21 (Múltiplos de 2 com um loop infinito)** Escreva um aplicativo que fique exibindo os múltiplos de 2 na janela de comando – ou seja, 2, 4, 8, 16, 32, 64 e assim por diante. Seu loop não deve terminar (isto é, deve ser um loop infinito). O que acontece quando esse programa é executado?

## Exercícios (seções C.14 a C.20)

- C.22** Descreva os quatro elementos básicos da repetição controlada por contador.
- C.23** (*Encontrar o menor valor*) Escreva um aplicativo que encontre o menor de vários valores inteiros. Suponha que o primeiro valor lido especifica o número de valores a serem inseridos pelo usuário.
- C.24** Suponha que  $i = 1$ ,  $j = 2$ ,  $k = 3$  e  $m = 2$ . O que cada uma das instruções a seguir imprime?
- `System.out.println( i == 1 );`
  - `System.out.println( j == 3 );`
  - `System.out.println( ( i >= 1 ) && ( j < 4 ) );`
  - `System.out.println( ( m <= 99 ) & ( k < m ) );`
  - `System.out.println( ( j >= i ) || ( k == m ) );`
  - `System.out.println( ( k + m < j ) | ( 3 - j >= k ) );`
  - `System.out.println( !( k > m ) );`

- C.25** (*Calculando o valor de  $\pi$* ) Calcule o valor de  $\pi$  a partir da série infinita

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \dots$$

Imprima uma tabela que mostre o valor de  $\pi$  aproximado pelo cálculo dos primeiros 200.000 termos dessa série. Quantos termos você precisa usar antes de obter um valor que comece com 3.14159?

- C.26** O que faz o trecho de programa a seguir?

```
for (i = 1; i <= 5; ++i)
{
 for (j = 1; j <= 3; ++j)
 {
 for (k = 1; k <= 4; ++k)
 System.out.print('*');

 System.out.println();
 } // fim do for interno

 System.out.println();
} // fim do for externo
```

- C.27** (*Música “The Twelve Days of Christmas”*) Escreva (o mais sucintamente possível) um aplicativo que use repetição e uma ou mais instruções `switch` para imprimir a música “The Twelve Days of Christmas”.

# Métodos: uma investigação mais aprofundada

# D

## Objetivos

Neste capítulo, você vai:

- Saber como os métodos e os campos estáticos são associados a classes, em vez de a objetos.
- Aprender como o mecanismo de chamada/retorno de método é suportado pela pilha de chamada de métodos.
- Saber como os pacotes agrupam classes relacionadas.
- Aprender a usar geração de números aleatórios para implementar aplicativos de jogos.
- Saber como a visibilidade das declarações é limitada a regiões específicas dos programas.
- Saber o que é sobrecarga de método e como criar métodos sobrecarregados.



- D.1** Introdução
- D.2** Módulos de programa em Java
- D.3** Métodos estáticos, campos estáticos e a classe `Math`
- D.4** Declaração de métodos com vários parâmetros
- D.5** Observações sobre declaração e uso de métodos
- D.6** Pilha de chamada de métodos e registros de ativação
- D.7** Promoção e conversão de argumentos
- D.8** Pacotes da API Java
- D.9** Introdução à geração de números aleatórios
  - D.9.1 Escala e deslocamento de números aleatórios
  - D.9.2 Repetitividade de números aleatórios para teste e depuração
- D.10** Estudo de caso: um jogo de azar introdução a enumerações
- D.11** Escopo das declarações
- D.12** Sobrecarga de métodos
- D.13** Para finalizar

*Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios*

## D.1 Introdução

Neste apêndice, estudamos os métodos com mais profundidade. Você vai ver que é possível chamar certos métodos, denominados métodos estáticos, sem a necessidade da existência de um objeto da classe. Vai aprender a declarar um método com mais de um parâmetro. Também vai aprender como a linguagem Java sabe qual método está em execução no momento, como as variáveis locais dos métodos são mantidas na memória e como um método sabe para onde retornar após concluir a execução.

Vamos nos desviar brevemente para as técnicas de simulação com geração de números aleatórios e vamos desenvolver uma versão de um jogo de dados de cassino, chamado *craps*, que usa a maior parte das técnicas de programação que você utilizou até este ponto no livro. Além disso, você vai aprender a declarar valores que não podem mudar (isto é, constantes) em seus programas.

Muitas das classes que você vai usar ao desenvolver aplicativos terão mais de um método com o mesmo nome. Essa técnica, denominada sobrecarga, é utilizada para implementar métodos que executam tarefas semelhantes para argumentos de diferentes tipos ou para diferentes números de argumentos.

## D.2 Módulos de programa em Java

Os programas Java são escritos combinando-se novos métodos e classes com os que já estão definidos na **Interface de Programação de Aplicativos Java** (também referida como **API Java** ou **biblioteca de classes Java**) e em várias outras bibliotecas de classe. Normalmente, as classes relacionadas são agrupadas em *pacotes* para que possam ser importadas para os programas e reutilizadas. Você vai aprender a agrupar suas próprias classes em pacotes no Apêndice F. A API Java fornece uma excelente coleção de classes predefinidas que contêm métodos para fazer cálculos matemáticos comuns, manipulações de string, manipulações de caractere, operações de entrada/saída, operações de banco de dados, operações de rede, processamento de arquivos, verificação de erros e muitas outras tarefas úteis.



### Observação sobre engenharia de software D.1

*Familiarize-se com a excelente coleção de classes e métodos fornecida pela API Java ([docs.oracle.com/javase/6/docs/api/](https://docs.oracle.com/javase/6/docs/api/)) e os reutilize quando possível. Isso reduz o tempo de desenvolvimento de programas e evita a introdução de erros de programação.*

Os métodos (chamados de **funções** ou **procedimentos** em algumas linguagens) ajudam a modularizar um programa, separando suas tarefas em unidades independentes. Você declarou métodos em todos os programas que escreveu. As instruções que ficam no corpo dos métodos são escritas apenas uma vez, ficam ocultas dos outros métodos e podem ser reutilizadas em vários lugares em um programa.

Um motivo para modularizar um programa com métodos é a estratégia de dividir para conquistar, a qual torna mais fácil manejar o desenvolvimento de programas, devido a sua construção a partir de partes menores e simples. Outro é a **reutilização de software** – usar métodos já existentes como elementos básicos para criar novos programas. Frequentemente, é possível criar programas principalmente a partir de métodos padronizados, em vez de desenvolver código personalizado. Por exemplo, em programas anteriores, não definimos como ler dados do teclado – a linguagem Java fornece esses recursos nos métodos da classe `Scanner`. Um terceiro motivo é evitar repetição de código. Dividir um programa em métodos significativos o torna mais fácil de ser depurado e mantido.

### D.3 Métodos estáticos, campos estáticos e a classe `Math`

Embora a maioria dos métodos seja executada em resposta a chamadas de método em *objetos específicos*, nem sempre esse é o caso. Às vezes, um método executa uma tarefa que não depende do conteúdo de nenhum objeto. Tal método se aplica à classe na qual é declarado como um todo e é conhecido como método estático ou **método de classe**. É normal as classes conterem métodos estáticos convenientes para executar tarefas comuns. Por exemplo, lembre-se de que usamos o método estático `pow` da classe `Math` para elevar um valor a uma potência, na Fig. C.15. Para declarar um método como estático, coloque a palavra-chave `static` antes do tipo de retorno na declaração do método. Para qualquer classe importada em seu programa, você pode chamar os métodos estáticos da classe especificando o nome da classe em que o método é declarado, seguido por um ponto (.) e pelo nome do método, como em

```
NomeDaClasse.nomeDoMétodo(argumentos)
```

Usamos vários métodos da classe `Math` aqui para apresentar a noção de métodos estáticos. A classe `Math` fornece um conjunto de métodos que permitem efetuar cálculos matemáticos comuns. Por exemplo, você pode calcular a raiz quadrada de `900.0` com a chamada de método estático

```
Math.sqrt(900.0)
```

A expressão anterior é avaliada como `30.0`. O método `sqrt` recebe um argumento de tipo `double` e retorna um resultado de tipo `double`. Para gerar o valor da chamada de método anterior na janela de comando, você poderia escrever a instrução

```
System.out.println(Math.sqrt(900.0));
```

Nessa instrução, o valor retornado por `sqrt` se torna o argumento do método `println`. Não há necessidade de criar um objeto `Math` antes de chamar o método `sqrt`. Além disso, *todos* os métodos da classe `Math` são estáticos – portanto, cada um é chamado precedendo-se seu nome com o nome da classe `Math` e o separador ponto (.).



#### Observação sobre engenharia de software D.2

A classe `Math` faz parte do pacote `java.lang`, o qual é importado implicitamente pelo compilador; portanto, não é necessário importar a classe `Math` para usar seus métodos.

Os argumentos dos métodos podem ser constantes, variáveis ou expressões. A Figura D.1 resume os métodos da classe `Math`. Na figura,  $x$  e  $y$  são de tipo `double`.

| Método                   | Descrição                                            | Exemplo                                                                                              |
|--------------------------|------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| <code>abs( x )</code>    | valor absoluto de $x$                                | <code>abs( 23.7 )</code> é 23.7<br><code>abs( 0.0 )</code> é 0.0<br><code>abs( -23.7 )</code> é 23.7 |
| <code>ceil( x )</code>   | arredonda $x$ para o menor inteiro não menor que $x$ | <code>ceil( 9.2 )</code> é 10.0<br><code>ceil( -9.8 )</code> é -9.0                                  |
| <code>cos( x )</code>    | cosseno trigonométrico de $x$ ( $x$ em radianos)     | <code>cos( 0.0 )</code> é 1.0                                                                        |
| <code>exp( x )</code>    | método exponencial $e^x$                             | <code>exp( 1.0 )</code> é 2.71828<br><code>exp( 2.0 )</code> é 7.38906                               |
| <code>floor( x )</code>  | arredonda $x$ para o maior inteiro não maior que $x$ | <code>floor( 9.2 )</code> é 9.0<br><code>floor( -9.8 )</code> é -10.0                                |
| <code>log( x )</code>    | logaritmo natural de $x$ (base $e$ )                 | <code>log( Math.E )</code> é 1.0<br><code>log( Math.E * Math.E )</code> é 2.0                        |
| <code>max( x, y )</code> | maior valor de $x$ e $y$                             | <code>max( 2.3, 12.7 )</code> é 12.7<br><code>max( -2.3, -12.7 )</code> é -2.3                       |
| <code>min( x, y )</code> | menor valor de $x$ e $y$                             | <code>min( 2.3, 12.7 )</code> é 2.3<br><code>min( -2.3, -12.7 )</code> é -12.7                       |
| <code>pow( x, y )</code> | $x$ elevado à potência $y$ (isto é, $x^y$ )          | <code>pow( 2.0, 7.0 )</code> é 128.0<br><code>pow( 9.0, 0.5 )</code> é 3.0                           |
| <code>sin( x )</code>    | seno trigonométrico de $x$ ( $x$ em radianos)        | <code>sin( 0.0 )</code> é 0.0                                                                        |
| <code>sqrt( x )</code>   | raiz quadrada de $x$                                 | <code>sqrt( 900.0 )</code> é 30.0                                                                    |
| <code>tan( x )</code>    | tangente trigonométrica de $x$ ( $x$ em radianos)    | <code>tan( 0.0 )</code> é 0.0                                                                        |

**Figura D.1** Métodos da classe `Math`.

### Constantes *PI* e *E* da classe *Math*

A classe `Math` declara dois campos que representam constantes matemáticas comumente usadas – `Math.PI` e `Math.E`. `Math.PI` (3.141592653589793) é a razão entre a circunferência de um círculo e seu diâmetro. `Math.E` (2.718281828459045) é o valor base dos logaritmos naturais (calculados com o método estático `log` de `Math`). Esses campos são declarados na classe `Math` com os modificadores `public`, `final` e `static`. Torná-los `public` permite a você usar esses campos em suas próprias classes. Todo campo declarado com a palavra-chave **final** é *constante* – seu valor não pode mudar depois que o campo é inicializado. `PI` e `E` são declarados com `final`, pois seus valores nunca mudam. Tornar esses campos estáticos permite que eles sejam acessados por meio do nome de classe `Math` e um separador ponto (`.`), exatamente como os métodos da classe `Math`. Lembre-se da seção B.4, que quando cada objeto de uma classe mantém sua própria cópia de um atributo, o campo que representa o atributo também é conhecido como variável de instância – cada objeto (instância) da classe tem uma instância separada da variável na memória. Existem campos para os quais cada objeto de uma classe *não* tem uma instância separada do campo. Esse é o caso dos campos estáticos, os quais também são conhecidos como **variáveis de classe**. Quando são criados objetos de uma classe que contém campos estáticos, todos os objetos dessa classe compartilham uma única cópia dos campos estáticos da classe. Juntas, as variáveis de classe (isto é, variáveis estáticas) e as variáveis de instância representam os campos de uma classe. Você vai aprender mais sobre campos estáticos na seção F.10.

**Por que o método *main* é declarado como *static*?**

Quando você executa a Java Virtual Machine (JVM) com o comando `java`, ela tenta chamar o método `main` da classe especificada – quando nenhum objeto da classe foi criado. Declarar `main` como `static` permite à JVM chamar `main` sem criar uma instância da classe. Ao executar seu aplicativo, você especifica seu nome de classe como argumento para o comando `java`, como em

```
java NomeDaClasse argumento1 argumento2 ...
```

A JVM carrega a classe especificada por *NomeDaClasse* e utiliza esse nome de classe para chamar o método `main`. No comando anterior, *NomeDaClasse* é um **argumento de linha de comando** para a JVM, que informa a classe a ser executada. Depois do *NomeDaClasse*, você também pode especificar uma lista de objetos `String` (separados por espaços) como argumentos de linha de comando que a JVM vai passar para seu aplicativo. Esses argumentos podem ser usados a fim de especificar opções (por exemplo, um nome de arquivo) para executar o aplicativo. Conforme você vai aprender no Apêndice E, seu aplicativo pode acessar esses argumentos de linha de comando e utilizá-los para personalização.

**D.4 Declaração de métodos com vários parâmetros**

Vamos considerar agora como escrever seus próprios métodos com *vários* parâmetros. A Figura D.2 usa um método chamado `maximum` para determinar e retornar o maior de três valores `double`. Em `main`, as linhas 14 a 18 solicitam ao usuário para que digite três valores `double` e, então, os leem. A linha 21 chama o método `maximum` (declarado nas linhas 28 a 41) para determinar o maior dos três valores que recebe como argumentos. Quando o método `maximum` retorna o resultado, na linha 21, o programa atribui o valor de retorno de `maximum` à variável local `result`. Então, a linha 24 gera o valor máximo na saída. No final desta seção, vamos discutir o uso do operador `+` na linha 24.

```

1 // Fig. D.2: MaximumFinder.java
2 // Método maximum declarado pelo programador com três parâmetros double.
3 import java.util.Scanner;
4
5 public class MaximumFinder
6 {
7 // obtém três valores de ponto flutuante e encontra o valor máximo
8 public static void main(String[] args)
9 {
10 // cria objeto Scanner da entrada da janela de comando
11 Scanner input = new Scanner(System.in);
12
13 // solicita e insere três valores de ponto flutuante
14 System.out.print(
15 "Enter three floating-point values separated by spaces: ");
16 double number1 = input.nextDouble(); // lê o primeiro double
17 double number2 = input.nextDouble(); // lê o segundo double
18 double number3 = input.nextDouble(); // lê o terceiro double
19
20 // determina o valor máximo
21 double result = maximum(number1, number2, number3);
22
23 // exibe o valor máximo
24 System.out.println("Maximum is: " + result);

```

**Figura D.2** Método `maximum` declarado pelo programador com três parâmetros `double`. (continua)



```

25 } // fim de main
26
27 // retorna o máximo de seus três parâmetros double
28 public static double maximum(double x, double y, double z)
29 {
30 double maximumValue = x; // presume que x é o maior, para começar
31
32 // determina se y é maior que maximumValue
33 if (y > maximumValue)
34 maximumValue = y;
35
36 // determina se z é maior que maximumValue
37 if (z > maximumValue)
38 maximumValue = z;
39
40 return maximumValue;
41 } // fim do método maximum
42 } // fim da classe MaximumFinder

```

Enter three floating-point values separated by spaces: **9.35 2.74 5.1**  
Maximum is: 9.35

Enter three floating-point values separated by spaces: **5.8 12.45 8.32**  
Maximum is: 12.45

Enter three floating-point values separated by spaces: **6.46 4.12 10.54**  
Maximum is: 10.54

**Figura D.2** Método `maximum` declarado pelo programador com três parâmetros `double`.

### ***As palavras-chave `public` e `static`***

A declaração de método `maximum` começa com a palavra-chave `public` para indicar que o método está “disponível para o público” – ele pode ser chamado a partir de métodos de outras classes. A palavra-chave `static` permite que o método `main` (outro método estático) chame `maximum` como mostrado na linha 21, sem qualificar o nome do método com o nome da classe `MaximumFinder` – os métodos estáticos da mesma classe podem chamar um ao outro diretamente. Qualquer outra classe que utilize `maximum` deve qualificar totalmente o nome do método com o nome da classe.

### ***Método `maximum`***

Na declaração de `maximum` (linhas 28 a 41), a linha 28 indica que ele retorna um valor `double`, que seu nome é `maximum` e que ele exige três parâmetros `double` (`x`, `y` e `z`) para cumprir sua tarefa. Vários parâmetros são especificados como uma lista separada por vírgulas. Quando `maximum` é chamado (linha 21), os parâmetros `x`, `y` e `z` são inicializados com os valores dos argumentos `number1`, `number2` e `number3`, respectivamente. Na chamada do método deve haver um argumento para cada parâmetro da declaração do método. Além disso, cada argumento deve ser *coerente* com o tipo do parâmetro correspondente. Por exemplo, um parâmetro `double` pode receber valores como 7.35, 22 ou -0.03456, mas não Strings como “hello” nem os valores booleanos `true` ou `false`.

Para determinar o valor máximo, começamos supondo que o parâmetro `x` contém o maior valor, de modo que a linha 30 declara a variável local `maximumValue` e a inicializa

com o valor do parâmetro `x`. Evidentemente, é possível que o parâmetro `y` ou `z` contenha o maior valor, de modo que precisamos comparar cada um desses valores com `maximumValue`. A instrução `if` nas linhas 33 e 34 determina se `y` é maior que `maximumValue`. Em caso positivo, a linha 34 atribui `y` a `maximumValue`. A instrução `if` nas linhas 37 e 38 determina se `z` é maior que `maximumValue`. Em caso positivo, a linha 38 atribui `z` a `maximumValue`. Nesse ponto, o maior dos três valores está em `maximumValue`, de modo que a linha 40 retorna esse valor para a linha 21. Quando o controle do programa volta para o ponto onde `maximum` foi chamado, os parâmetros `x`, `y` e `z` de `maximum` não existem mais na memória.



### Observação sobre engenharia de software D.3

*As variáveis devem ser declaradas como campos somente se forem exigidas para uso em mais de um método da classe ou se o programa precisa salvar seus valores entre chamadas para os métodos da classe.*

### Implementação do método `maximum` reutilizando o método `Math.max`

O corpo inteiro de nosso método `maximum` também poderia ser implementado com duas chamadas para `Math.max`, como segue:

```
return Math.max(x, Math.max(y, z));
```

A primeira chamada de `Math.max` especifica os argumentos `x` e `Math.max( y, z )`. Antes que qualquer método possa ser chamado, seus argumentos devem ser avaliados para se determinar seus valores. Se um argumento é uma chamada de método, essa deve ser feita para determinar seu valor de retorno. Assim, na instrução anterior, `Math.max( y, z )` é avaliado para determinar o máximo entre `y` e `z`. Então, o resultado é passado como segundo argumento para a outra chamada de `Math.max`, o qual retorna o maior de seus dois argumentos.

### Montando strings com concatenação de strings

A linguagem Java permite agrupar objetos `String` em strings maiores, usando os operadores `+` ou `+=`. Isso é conhecido como **concatenação de strings**. Quando os dois operandos do operador `+` são objetos `String`, o operador `+` cria um novo objeto `String` no qual os caracteres do operando da direita são colocados no final dos do operando da esquerda – por exemplo, a expressão `"hello " + "there"` cria a `String` `"hello there"`.

Na linha 24 da Fig. D.2, a expressão `"Maximum is: " + result` usa o operador `+` com operandos de tipos `String` e `double`. *Todo valor e objeto primitivo em Java tem uma representação String*. Quando um dos operandos do operador `+` é `String`, o outro é convertido em `String` e os dois são *concatenados*. Na linha 24, o valor `double` é convertido em sua representação `String` e colocado no final da `String` `"Maximum is: "`. Se houver zeros à direita em um valor `double`, eles serão *descartados* quando o número for convertido em `String` – por exemplo 9.3500 seria representado como 9.35.

Os valores primitivos usados na concatenação de `String` são convertidos em `String`. Um valor booleano concatenado com uma `String` é convertido na `String` `"true"` ou `"false"`. Todos os objetos têm um método `toString` que retorna uma representação `String` do objeto. Quando um objeto é concatenado com uma `String`, o método `toString` do objeto é chamado implicitamente para obter sua representação `String`. `ToString` pode ser chamado explicitamente.



### Erro de programação comum D.1

É um erro de sintaxe dividir uma literal String por várias linhas. Se necessário, você pode dividir uma String em várias Strings menores e usar concatenação para formar a String desejada.



### Erro de programação comum D.2

Confundir o operador + usado para concatenação de string com o operador + usado para adição pode levar a resultados estranhos. A linguagem Java avalia os operandos de um operador da esquerda para a direita. Por exemplo, se a variável inteira y tem o valor 5, a expressão "y + 2 = " + y + 2 resulta na string "y + 2 = 52" e não em "y + 2 = 7", pois o primeiro valor de y (5) é concatenado com a string "y + 2 = " e, então, o valor 2 é concatenado com a nova string maior "y + 2 = 5". A expressão "y + 2 = " + (y + 2) produz o resultado desejado "y + 2 = 7".

## D.5 Observações sobre declaração e uso de métodos

Existem três maneiras de chamar um método:

1. Usando um nome de método sozinho para chamar outro método da *mesma* classe – como `maximum(number1, number2, number3)` na linha 21 da Fig. D.2.
2. Usando uma variável que contenha uma referência para um objeto, seguida de um ponto (.) e do nome do método para chamar um método não estático do objeto referenciado – como a chamada de método na linha 13 da Fig. C.3, `myGradeBook.displayMessage()`, que chama um método da classe `GradeBook` a partir do método `main` de `GradeBookTest`.
3. Usando o nome da classe e um ponto (.) para chamar um método estático de uma classe – como `Math.sqrt(900.0)` na seção D.3.

Um método estático pode chamar diretamente *apenas* outros métodos estáticos da mesma classe (isto é, usando o nome do método sozinho) e pode manipular diretamente *apenas* variáveis estáticas da mesma classe. Para acessar os membros não estáticos da classe, um método estático deve usar uma referência para um objeto da classe. Muitos objetos de uma classe, cada um com suas próprias cópias das variáveis de instância, podem existir ao mesmo tempo. Suponha que um método estático chamasse um método não estático diretamente. Como ele saberia quais variáveis de instância do objeto deveria manipular? O que aconteceria se nenhum objeto da classe existisse no momento em que o método não estático fosse chamado? Por isso, a linguagem Java não permite que um método estático acesse diretamente membros não estáticos da mesma classe.

Existem três maneiras de retornar o controle para a instrução que chama um método. Se o método não retorna um resultado, o controle é retornado quando o fluxo do programa atinge a chave de fechamento do método ou quando a instrução

```
return;
```

é executada. Se o método retorna um resultado, a instrução

```
return expressão;
```

avalia a *expressão* e, então, retorna o resultado para o chamador.



### Erro de programação comum D.3

*Declarar um método fora do corpo de uma declaração de classe ou dentro do corpo de outro método é um erro de sintaxe.*



### Erro de programação comum D.4

*Redeclarar um parâmetro como uma variável local no corpo do método causa um erro de compilação.*

## D.6 Pilha de chamada de métodos e registros de ativação

Para entender como a linguagem Java faz chamadas de método, primeiro precisamos considerar uma estrutura de dados (isto é, coleção de itens de dados relacionados) conhecida como **pilha**. Você pode imaginar uma pilha como um empilhamento de pratos. Quando um prato é colocado na pilha, isso normalmente acontece no topo (o que é referido como **colocar** [push] o prato na pilha). Do mesmo modo, quando um prato é removido da pilha, isso sempre acontece a partir do topo (o que é referido como **retirar** [pop] o prato da pilha). As pilhas são conhecidas como **estruturas de dados LIFO (sigla em inglês para last-in, first-out)** – o último item colocado (inserido) na pilha é o primeiro item retirado (removido) da pilha.

Quando um programa chama um método, o método chamado deve saber como retornar para seu chamador, de modo que o endereço de retorno do método que está chamando é colocado na **pilha de execução do programa** (às vezes referida como **pilha de chamada de método**). Se ocorre uma série de chamadas de método, os sucessivos endereços de retorno são colocados na pilha, na ordem último a entrar, primeiro a sair, para que cada método possa retornar para seu chamador.

A pilha de execução do programa também contém a memória para as variáveis locais utilizadas em cada chamada de um método durante a execução de um programa. Esses dados, armazenados como parte da pilha de execução do programa, são conhecidos como **registro de ativação** ou **stack frame** da chamada de método. Quando uma chamada de método é feita, o registro de ativação desse método é colocado na pilha de execução do programa. Quando o método retorna para seu chamador, seu registro de ativação é retirado da pilha e aquelas variáveis locais não são mais conhecidas pelo programa. Se uma variável local contendo uma referência para um object é a única variável no programa com uma referência para esse objeto, então, quando o registro de ativação que contém essa variável local é retirado da pilha, o objeto não pode mais ser acessado pelo programa e finalmente será excluído da memória pela JVM durante a “coleta de lixo”. Discutimos a coleta de lixo na seção F.9.

Evidentemente, a memória de um computador é finita; portanto, apenas certa quantidade pode ser usada para armazenar registros de ativação na pilha de execução do programa. Se existe mais de uma chamada de método que tenha seus registros de ativação armazenados, ocorre um erro conhecido como **estouro da pilha (stack overflow)**.

## D.7 Promoção e conversão de argumentos

Outra característica importante das chamadas de método é a **promoção de argumentos** – converter o valor de um argumento, se possível, no tipo que o método espera receber

em seu parâmetro correspondente. Por exemplo, um programa pode chamar o método `sqrt` de `Math` com um argumento `int`, mesmo sendo esperado um argumento `double`. A instrução

```
System.out.println(Math.sqrt(4));
```

avalia `Math.sqrt(4)` corretamente e imprime o valor `2.0`. A lista de parâmetros da declaração do método faz a linguagem Java converter o valor `int` `4` para o valor `double` `4.0`, antes de passá-lo para o método `sqrt`. Tais conversões podem levar a erros de compilação, caso as **regras de promoção** do Java não sejam satisfeitas. Essas regras especificam as conversões permitidas – isto é, quais podem ser realizadas sem perda de dados. No exemplo do método `sqrt` anterior, um valor `int` é convertido em um valor `double` sem alterar o valor. Contudo, converter um valor `double` em um valor `int` trunca a parte fracionária do valor `double` – assim, parte do valor é perdida. Converter tipos inteiros grandes para tipos inteiros pequenos (por exemplo, `long` para `int` ou `int` para `short`) também pode resultar em valores alterados.

As regras de promoção se aplicam a expressões contendo valores de dois ou mais tipos primitivos e a valores de tipo primitivo passados como argumentos para métodos. Cada valor é promovido para o tipo “mais alto” na expressão. Na verdade, a expressão usa uma cópia temporária de cada valor – os tipos dos valores originais permanecem intactos. A Figura D.3 lista os tipos primitivos e os tipos para os quais cada um pode ser promovido. As promoções válidas para determinado tipo são sempre para um tipo mais alto na tabela. Por exemplo, um valor `int` pode ser promovido para os tipos mais altos `long`, `float` e `double`.

| Tipo                 | Promoções válidas                                                                                                                  |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <code>double</code>  | Nenhuma                                                                                                                            |
| <code>float</code>   | <code>double</code>                                                                                                                |
| <code>long</code>    | <code>float</code> ou <code>double</code>                                                                                          |
| <code>int</code>     | <code>long</code> , <code>float</code> ou <code>double</code>                                                                      |
| <code>char</code>    | <code>int</code> , <code>long</code> , <code>float</code> ou <code>double</code>                                                   |
| <code>short</code>   | <code>int</code> , <code>long</code> , <code>float</code> ou <code>double</code> (mas não <code>char</code> )                      |
| <code>byte</code>    | <code>short</code> , <code>int</code> , <code>long</code> , <code>float</code> ou <code>double</code> (mas não <code>char</code> ) |
| <code>boolean</code> | Nenhum (os valores booleanos não são considerados números em Java)                                                                 |

**Figura D.3** Promoções permitidas para tipos primitivos.

Caso o tipo inferior não possa representar o valor do tipo superior, a conversão de valores para tipos mais abaixo na tabela da Fig. D.3 resulta em valores diferentes (por exemplo, o valor `int` `2000000` não pode ser representado como `short`, e nenhum número de ponto flutuante com algarismos após o ponto decimal pode ser representado em um tipo inteiro, como `long`, `int` ou `short`). Portanto, nos casos em que possa haver perda de informação devido à conversão, o compilador Java exige que você use um operador de conversão (apresentado na seção C.9) para forçar explicitamente a conversão – caso contrário, ocorrerá um erro de compilação. Isso permite que você “assuma o controle” do compilador. Basicamente, você diz, “Eu sei que essa conversão pode causar perda de informação, mas para meus propósitos aqui, tudo bem”. Suponha que o método `square` calcule o quadrado de um valor inteiro e, portanto, exija um argumento `int`. Para

chamar `square` com um argumento `double` chamado `doubleValue`, seríamos obrigados a escrever a chamada de método como

```
square((int) doubleValue)
```

Essa chamada converte explicitamente uma *cópia* do valor da variável `doubleValue` em um inteiro para uso no método `square`. Assim, se o valor de `doubleValue` é 4.5, o método recebe o valor 4 e retorna 16, não 20.25.

## D.8 Pacotes da API Java

Como você viu, a linguagem Java contém muitas classes predefinidas agrupadas em categorias de classes relacionadas, chamadas pacotes. Juntos, eles são conhecidos como Interface de Programação de Aplicativo Java (API Java) ou biblioteca de classes Java. Uma grande vantagem da linguagem Java são os milhares de classes da API Java. Alguns pacotes importantes da API Java utilizados nos apêndices deste livro estão descritos na Fig. D.4, os quais representam apenas uma pequena parte dos componentes reutilizáveis da API.

| Pacote                            | Descrição                                                                                                                                                                                                                                                     |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>java.awt.event</code>       | O <b>Java Abstract Window Toolkit Event Package</b> contém classes e interfaces que permitem o tratamento de eventos para componentes de interface gráfica do usuário nos pacotes <code>java.awt</code> e <code>javax.swing</code> .                          |
| <code>java.io</code>              | O <b>Java Input/Output Package</b> contém classes e interfaces que permitem aos programas inserir e gerar dados na saída.                                                                                                                                     |
| <code>java.lang</code>            | O <b>Java Language Package</b> contém classes e interfaces (discutidas por todo o livro) exigidas por muitos programas Java. Esse pacote é importado pelo compilador para todos os programas.                                                                 |
| <code>java.util</code>            | O <b>Java Utilities Package</b> contém classes utilitárias e interfaces que permitem ações como manipulações de data e hora, processamento de números aleatórios (classe <code>Random</code> ) e o armazenamento e processamento de grandes volumes de dados. |
| <code>java.util.concurrent</code> | O <b>Java Concurrency Package</b> contém classes utilitárias e interfaces para implementar programas que podem executar várias tarefas em paralelo.                                                                                                           |
| <code>javax.swing</code>          | O <b>Java Swing GUI Components Package</b> contém classes e interfaces para componentes de interface gráfica do usuário Swing do Java, que fornecem suporte para interfaces portáteis.                                                                        |

**Figura D.4** Pacotes da API Java (um subconjunto).

O conjunto de pacotes disponíveis em Java é muito grande. Além dos que estão resumidos na Fig. D.4, o Java contém pacotes para elementos gráficos complexos, interfaces gráficas do usuário avançadas, impressão, interligação em rede avançada, segurança, processamento de banco de dados, multimídia, acessibilidade (para deficientes), programação concorrente, criptografia, processamento de XML e muitas outras capacidades. Muitos outros pacotes também estão disponíveis para download no endereço `java.sun.com`.

Mais informações sobre os métodos predefinidos de uma classe Java podem ser encontradas na documentação da API Java, no endereço `docs.oracle.com/javase/6/docs/api/`. Quando visitar esse site, clique no link **Index** para ver uma listagem em ordem alfabética

de todas as classes e métodos da API Java. Localize o nome da classe e clique em seu link para ver a descrição online da classe. Clique no link **METHOD** para ver uma tabela dos métodos da classe. Cada método estático será listado com a palavra “static” precedendo seu tipo de retorno.

## D.9 Introdução à geração de números aleatórios

Agora, mudaremos um pouco de assunto para falarmos sobre um tipo popular de aplicação da programação – simulação e jogos. Nesta seção e na seguinte, desenvolvemos um programa de jogo muito bem estruturado, com vários métodos. O programa utiliza a maior parte das instruções de controle apresentadas até aqui nos apêndices e introduz vários conceitos de programação novos.

Números aleatórios podem ser introduzidos em um programa por meio de um objeto da classe **Random** (pacote `java.util`) ou por meio do método estático `random` da classe **Math**. Um objeto **Random** pode produzir valores aleatórios `boolean`, `byte`, `float`, `double`, `int`, `long` e Gaussianos, ao passo que o método `random` de **Math** só pode produzir valores `double` no intervalo  $0.0 \leq x < 1.0$ , onde  $x$  é o valor retornado pelo método `random`. Nos próximos exemplos, usamos objetos da classe **Random** para produzir valores aleatórios. Discutimos aqui somente os valores aleatórios `int`. Para obter mais informações sobre a classe **Random**, consulte [docs.oracle.com/javase/6/docs/api/java/util/Random.html](https://docs.oracle.com/javase/6/docs/api/java/util/Random.html). Um objeto gerador de números aleatórios pode ser criado como segue:

```
Random randomNumbers = new Random();
```

Considere a instrução a seguir:

```
int randomValue = randomNumbers.nextInt();
```

O método `nextInt` de **Random** gera um valor `int` aleatório no intervalo de -2.147.483.648 a +2.147.483.647, inclusive. Se ele realmente produz valores aleatoriamente, então todo valor no intervalo deve ter uma chance (ou probabilidade) igual de ser escolhido sempre que `nextInt` é chamado. Na verdade, os números são **pseudoaleatórios** – uma sequência de valores produzidos por um cálculo matemático complexo. O cálculo usa a hora atual do dia (a qual, é claro, muda constantemente) para **semear** o gerador de números aleatórios, de modo que cada execução de um programa produz uma sequência diferente de valores aleatórios.

O intervalo de valores produzidos diretamente pelo método `nextInt` em geral difere do intervalo exigido por um aplicativo Java específico. Por exemplo, um programa que simula o lançamento de uma moeda pode exigir apenas 0 para “cara” e 1 para “coroa”. Um programa que simula o lançamento de um dado de seis faces poderia exigir inteiros aleatórios no intervalo de 1 a 6. Um programa que prevê aleatoriamente o próximo tipo de nave espacial (dentre quatro possibilidades) que vai voar no horizonte em um videogame poderia exigir inteiros aleatórios no intervalo de 1 a 4. Para casos como esses, a classe **Random** fornece outra versão do método `nextInt`, que recebe um argumento `int` e retorna um valor de 0 até, mas não incluindo, o valor do argumento. Por exemplo, para lançamento de moeda, a instrução a seguir retorna 0 ou 1.

```
int randomValue = randomNumbers.nextInt(2);
```

### D.9.1 Escala e deslocamento de números aleatórios

Para demonstrar os números aleatórios, vamos mostrar a simulação do lançamento de um dado de seis faces. Começamos usando `nextInt` para produzir valores aleatórios no intervalo de 0 a 5, como segue:

```
face = randomNumbers.nextInt(6);
```

O argumento 6 – chamado de **fator de escala** – representa o número de valores únicos que `nextInt` deve produzir (neste caso, seis: 0, 1, 2, 3, 4 e 5). Essa manipulação é chamada de **escala** do intervalo de valores produzidos pelo método `nextInt` de `Random`. Um dado de seis faces tem os números de 1 a 6 em seus lados, não de 0 a 5. Assim, **deslocamos** o intervalo de números produzidos adicionando um **valor de deslocamento** – neste caso, 1 – em nosso resultado anterior, como em

```
face = 1 + randomNumbers.nextInt(6);
```

O valor de deslocamento (1) especifica o *primeiro* valor no intervalo desejado de inteiros aleatórios. A instrução anterior atribui a `face` um valor inteiro aleatório no intervalo de 1 a 6. Os números produzidos por `nextInt` ocorrem com probabilidade aproximadamente igual.

#### Generalizando os cálculos de números aleatórios

A instrução anterior sempre atribui à variável `face` um valor inteiro no intervalo  $1 \leq \text{face} \leq 6$ . A amplitude desse intervalo (isto é, o número de valores inteiros consecutivos no intervalo) é 6, e o número inicial do intervalo é 1. A amplitude do intervalo é determinada pelo número 6 passado como argumento para o método `nextInt` de `Random`, e o número inicial é o número 1 adicionado ao resultado da chamada de `nextInt`. Podemos generalizar esse resultado como

```
number = valorDeDeslocamento + randomNumbers.nextInt(fatorDeEscala);
```

onde *valorDeDeslocamento* especifica o primeiro número no intervalo desejado de inteiros consecutivos e *fatorDeEscala* especifica quantos números há no intervalo.

Também é possível escolher inteiros aleatoriamente a partir de conjuntos de valores que não sejam consecutivos. Por exemplo, para obter um valor aleatório da sequência 2, 5, 8, 11 e 14, você poderia usar a instrução

```
number = 2 + 3 * randomNumbers.nextInt(5);
```

Nesse caso, `randomNumbers.nextInt(5)` produz valores no intervalo de 0 a 4. Cada valor produzido é multiplicado por 3 para gerar um número na sequência 0, 3, 6, 9 e 12. Somamos 2 a esse valor para deslocar o intervalo de valores e obter um valor da sequência 2, 5, 8, 11 e 14. Podemos generalizar esse resultado como

```
number = valorDeDeslocamento +
diferençaEntreOsValores * randomNumbers.nextInt(fatorDeEscala);
```

onde *valorDeDeslocamento* especifica o primeiro número no intervalo de valores desejado, *diferençaEntreOsValores* representa a diferença constante entre números consecutivos na sequência e *fatorDeEscala* especifica quantos números existem no intervalo.



### D.9.2 Repetitividade de números aleatórios para teste e depuração

Os métodos da classe `Random` geram, na verdade, números pseudoaleatórios baseados em cálculos matemáticos complexos – a sequência de números parece ser aleatória. O cálculo que produz os números usa a hora do dia como **valor semente** para mudar o ponto de partida da sequência. Cada novo objeto `Random` semeia-se a si próprio com um valor baseado no relógio de sistema do computador no momento em que o objeto é criado, permitindo que cada execução de um programa produza uma sequência de números aleatórios diferente.

Ao se depurar um aplicativo, muitas vezes é útil repetir exatamente a mesma sequência de números pseudoaleatórios durante cada execução do programa. Essa repetitividade permite a você comprovar que seu aplicativo está funcionando para uma sequência específica de números aleatórios antes de testá-lo com sequências diferentes. Quando a repetitividade for importante, crie um objeto `Random` como segue:

```
Random randomNumbers = new Random(seedValue);
```

O argumento `seedValue` (de tipo `long`) semeia o cálculo de números aleatórios. Se o mesmo `seedValue` é usado a cada vez, o objeto `Random` produz a mesma sequência de números. Você pode configurar a semente de um objeto `Random` a qualquer momento durante a execução do programa, chamando o método `set` do objeto, como em

```
randomNumbers.set(seedValue);
```



#### Dica de prevenção de erro D.1

*Ao desenvolver um programa, crie o objeto `Random` com um valor semente específico, a fim de produzir uma sequência de números que possa ser repetida cada vez que o programa for executado. Se ocorrer um erro de lógica, corrija-o e teste o programa novamente com o mesmo valor semente – isso permite reconstruir a mesma sequência de números que causou o erro. Uma vez removido o erro de lógica, crie o objeto `Random` sem usar um valor semente, fazendo o objeto `Random` gerar uma nova sequência de números cada vez que o programa for executado.*

## D.10 Estudo de caso: um jogo de azar – introdução a enumerações

Um jogo de azar popular é o jogo de dados conhecido como *craps*, o qual é jogado em cassinos e becos em todo o mundo. As regras do jogo são simples:

*Você lança dois dados. Cada dado tem seis faces, as quais contêm um, dois, três, quatro, cinco e seis pontos, respectivamente. Depois que os dados param, é calculada a soma dos pontos nas duas faces que estão para cima. Se a soma for 7 ou 11 no primeiro lançamento, você vence. Se a soma for 2, 3 ou 12 no primeiro lançamento (chamado de “craps”), você perde (isto é, a “banca” vence). Se a soma for 4, 5, 6, 7, 8, 9 ou 10 no primeiro lançamento, essa soma se torna seu “ponto”. Para vencer, você deve continuar a lançar os dados até “fazer o ponto” (isto é, rolar o mesmo valor do ponto). Você perde rolando um 7 antes de fazer o ponto.*

A Figura D.5 simula o jogo de *craps* usando métodos para implementar a lógica do jogo. O método `main` (linhas 21 a 65) chama o método `rollDice` (linhas 68 a 81) conforme for necessário, para rolar os dados e calcular a soma. Os exemplos de saída mostram vitória e derrota no primeiro lançamento, e vitória e derrota em um lançamento subsequente.

```

1 // Fig. D.5: Craps.java
2 // Classe Craps que simula o jogo de dados craps.
3 import java.util.Random;
4
5 public class Craps
6 {
7 // cria gerador de números aleatórios para usar no método rollDice
8 private static final Random randomNumbers = new Random();
9
10 // enumeração com constantes que representam o status do jogo
11 private enum Status { CONTINUE, WON, LOST };
12
13 // constantes que representam lançamentos comuns do dado
14 private static final int SNAKE_EYES = 2;
15 private static final int TREY = 3;
16 private static final int SEVEN = 7;
17 private static final int YO_LEVEN = 11;
18 private static final int BOX_CARS = 12;
19
20 // joga um jogo de craps
21 public static void main(String[] args)
22 {
23 int myPoint = 0; // ponto se não houve vitória ou derrota no primeiro lançamento
24 Status gameStatus; // pode conter CONTINUE, WON ou LOST
25
26 int sumOfDice = rollDice(); // primeiro lançamento dos dados
27
28 // determina o status do jogo e o ponto com base no primeiro lançamento
29 switch (sumOfDice)
30 {
31 case SEVEN: // vence com 7 no primeiro lançamento
32 case YO_LEVEN: // vence com 11 no primeiro lançamento
33 gameStatus = Status.WON;
34 break;
35 case SNAKE_EYES: // perde com 2 no primeiro lançamento
36 case TREY: // perde com 3 no primeiro lançamento
37 case BOX_CARS: // perde com 12 no primeiro lançamento
38 gameStatus = Status.LOST;
39 break;
40 default: // não venceu nem perdeu; portanto, lembra o ponto
41 gameStatus = Status.CONTINUE; // o jogo não terminou
42 myPoint = sumOfDice; // lembra o ponto
43 System.out.printf("Point is %d\n", myPoint);
44 break; // opcional no final do switch
45 } // fim do switch
46
47 // enquanto o jogo não termina
48 while (gameStatus == Status.CONTINUE) // nem WON nem LOST
49 {
50 sumOfDice = rollDice(); // lança os dados novamente
51
52 // determine game status
53 if (sumOfDice == myPoint) // vence fazendo o ponto
54 gameStatus = Status.WON;
55 else

```

**Figura D.5** Classe Craps que simula o jogo de dados *craps*. (continua)

```

56 if (sumOfDice == SEVEN) // perde lançando 7 antes do ponto
57 gameStatus = Status.LOST;
58 } // fim do while
59
60 // exibe mensagem de vitória ou derrota
61 if (gameStatus == Status.WON)
62 System.out.println("Player wins");
63 else
64 System.out.println("Player loses");
65 } // fim de main
66
67 // lança os dados, calcula a soma e exibe os resultados
68 public static int rollDice()
69 {
70 // escolhe valores aleatórios para o dado
71 int die1 = 1 + randomNumbers.nextInt(6); // lançamento do primeiro dado
72 int die2 = 1 + randomNumbers.nextInt(6); // lançamento do segundo dado
73
74 int sum = die1 + die2; // soma dos valores dos dados
75
76 // exibe os resultados desse lançamento
77 System.out.printf("Player rolled %d + %d = %d\n",
78 die1, die2, sum);
79
80 return sum; // retorna a soma dos dados
81 } // fim do método rollDice
82 } // fim da classe Craps

```

```

Player rolled 5 + 6 = 11
Player wins

```

```

Player rolled 5 + 4 = 9
Point is 9
Player rolled 4 + 2 = 6
Player rolled 3 + 6 = 9
Player wins

```

```

Player rolled 1 + 2 = 3
Player loses

```

```

Player rolled 2 + 6 = 8
Point is 8
Player rolled 5 + 1 = 6
Player rolled 2 + 1 = 3
Player rolled 1 + 6 = 7
Player loses

```

**Figura D.5** Classe Craps que simula o jogo de dados *craps*.

### **Método `rollDice`**

De acordo com as regras do jogo, o jogador deve lançar dois dados na primeira vez e deve fazer o mesmo em todos os lançamentos subsequentes. Declaramos o método `rollDice` (Fig. D.5, linhas 68 a 81) para lançar os dados, calcular e imprimir sua soma. O método `rollDice` é declarado uma vez, mas é chamado em dois lugares (linhas 26 e 50) em `main`, que contém a lógica para um jogo de *craps* completo. O método `rollDice`

não recebe argumentos, de modo que tem uma lista de parâmetros vazia. Sempre que é chamado, `rollDice` retorna a soma dos dados; portanto, o tipo de retorno `int` é indicado no cabeçalho do método (linha 68). Embora as linhas 71 e 72 pareçam iguais (a não ser pelos nomes dos dados), elas não produzem necessariamente o mesmo resultado. Cada uma dessas instruções produz um valor aleatório no intervalo de 1 a 6. A variável `randomNumbers` (usada nas linhas 71 e 72) *não* é declarada no método. Em vez disso, é declarada como uma variável `private static final` da classe e inicializada na linha 8. Isso nos permite criar um objeto `Random` que é reutilizado em cada chamada de `rollDice`. Se houvesse um programa que contivesse várias instâncias da classe `Craps`, todas compartilhariam esse único objeto `Random`.

### Variáveis locais do método `main`

O jogo é razoavelmente complicado. O jogador pode vencer ou perder no primeiro lançamento, ou pode vencer ou perder em qualquer lançamento subsequente. O método `main` (linhas 21 a 65) usa a variável local `myPoint` (linha 23) para armazenar o “ponto” para o caso de o jogador não vencer ou perder no primeiro lançamento, a variável local `gameStatus` (linha 24) para monitorar o status global do jogo e a variável local `sumOfDice` (linha 26) para armazenar a soma dos dados para o lançamento mais recente. A variável `myPoint` é inicializada com 0 para garantir que o aplicativo seja compilado. Se você não inicializar `myPoint`, o compilador emitirá um erro, pois `myPoint` não recebeu um valor em *cada* case da instrução `switch` e, assim, o programa poderia tentar usar `myPoint` antes que recebesse um valor. Em contraste, `gameStatus` *recebe* um valor em *cada* case da instrução `switch` – assim, há garantia de que é inicializada antes de ser usada e não precisa ser inicializada.

### Status do tipo `enum`

A variável local `gameStatus` (linha 24) é declarada com um novo tipo chamado `Status` (declarado na linha 11). O tipo `Status` é um membro `private` da classe `Craps`, pois vai ser usado somente nessa classe. `Status` é um tipo chamado **enumeração**, o qual, em sua forma mais simples, declara um conjunto de constantes representadas por identificadores. Uma enumeração é um tipo especial de classe, que é introduzida pela palavra-chave **`enum`** e um nome de tipo (neste caso, `Status`). Assim como nas classes, chaves delimitam o corpo de uma declaração `enum`. Dentro das chaves, existe uma lista separada por vírgulas de **constantes de enumeração**, cada uma representando um valor único. Os identificadores em `enum` devem ser exclusivos. Você vai aprender mais sobre enumerações no Apêndice F.



#### Boa prática de programação D.1

*É uma convenção usar apenas letras maiúsculas nos nomes de constantes de enumeração. Isso as faz se destacar e lembrar que não são variáveis.*

As variáveis de tipo `Status` só podem receber as três constantes declaradas na enumeração (linha 11), senão ocorrerá um erro de compilação. Quando se vence no jogo, o programa configura a variável local `gameStatus` como `Status.WON` (linhas 33 e 54). Quando se perde no jogo, o programa configura a variável local `gameStatus` como `Status.LOST` (linhas 38 e 57). Caso contrário, o programa configura a variável local `gameStatus` como `Status.CONTINUE` (linha 41) para indicar que o jogo não terminou e os dados devem ser lançados novamente.



### Boa prática de programação D.2

*Usar constantes de enumeração (como `Status.WON`, `Status.LOST` e `Status.CONTINUE`), em vez de valores literais (como 0, 1 e 2), torna os programas mais fáceis de ler e manter.*

### Lógica do método `main`

A linha 26 em `main` chama `rollDice`, o qual escolhe dois valores aleatórios de 1 a 6, exibe o valor do primeiro dado, do segundo dado e a soma deles, e retorna a soma. Em seguida, o método `main` entra na instrução `switch` (linhas 29 a 45), a qual utiliza o valor de `sumOfDice` da linha 26 para determinar se o jogo foi vencido ou perdido, ou se deve continuar com outro lançamento. Os valores resultantes de uma vitória ou derrota no primeiro lançamento são declarados como constantes `private static final int` nas linhas 14 a 18. Os nomes de identificador usam o jargão dos cassinos (em inglês) para essas somas. Essas constantes, assim como as constantes `enum`, são declaradas com todas as letras maiúsculas por convenção, para fazê-las se destacar no programa. As linhas 31 a 34 determinam se o jogador ganhou no primeiro lançamento com `SEVEN` (7) ou `YO_LEVEN` (11). As linhas 35 a 39 determinam se ele perdeu no primeiro lançamento com `SNAKE_EYES` (2), `TREY` (3) ou `BOX_CARS` (12). Após o primeiro lançamento, se o jogo não terminou, o `case default` (linhas 40 a 44) configura `gameStatus` como `Status.CONTINUE`, salva `sumOfDice` em `myPoint` e exibe o ponto.

Se ainda estamos tentando “fazer nosso ponto” (isto é, se o jogo está continuando a partir de um lançamento anterior), as linhas 48 a 58 são executadas. A linha 50 lança os dados novamente. Se `sumOfDice` corresponde a `myPoint` (linha 53), a linha 54 configura `gameStatus` como `Status.WON` e, então, o loop termina, pois o jogo acabou. Se `sumOfDice` é `SEVEN` (linha 56), a linha 57 configura `gameStatus` como `Status.LOST` e o loop termina, pois o jogo acabou. Quando o jogo acaba, as linhas 61 a 64 exibem uma mensagem indicando se o jogador ganhou ou perdeu, e o programa termina.

O programa usa vários mecanismos de controle já discutidos. A classe `Craps` utiliza dois métodos – `main` e `rollDice` (chamado duas vezes a partir de `main`) – e as instruções de controle `switch`, `while`, `if...else` e `if` aninhada. Observe também o uso de vários rótulos `case` na instrução `switch`, para executar as mesmas instruções para somas de `SEVEN` e `YO_LEVEN` (linhas 31 e 32) e para somas de `SNAKE_EYES`, `TREY` e `BOX_CARS` (linhas 35 a 37).

### Por que algumas constantes não são definidas como `enum`

Talvez você esteja se perguntando por que declaramos as somas dos dados como constantes `private static final int`, em vez de `enum`. O motivo é que o programa precisa comparar a variável `int sumOfDice` (linha 26) com essas constantes para determinar o resultado de cada lançamento. Suponha que tivéssemos declarado a `enum Sum` contendo constantes (por exemplo, `Sum.SNAKE_EYES`) representando as cinco somas utilizadas no jogo e, depois, usado essas constantes na instrução `switch` (linhas 29 a 45). Isso nos impediria de usar `sumOfDice` como expressão de controle da instrução `switch`, pois a linguagem Java *não* permite que um valor `int` seja comparado com uma constante de enumeração. Para obtermos a mesma funcionalidade do programa atual, precisaríamos usar uma variável `currentSum` de tipo `Sum` como expressão de controle do `switch`. Infelizmente, a linguagem Java não oferece um modo fácil de converter um valor `int` em uma constante `enum` em particular. Isso poderia ser feito com uma instrução `switch` separada. Claramente, seria trabalhoso e não melhoraria a clareza do programa (anulando assim o propósito de usar `enum`).

## D.11 Escopo das declarações

Você já viu declarações de várias entidades do Java, como classes, métodos, variáveis e parâmetros. As declarações introduzem nomes que podem ser usados para fazer referência a essas entidades. O **escopo** de uma declaração é a parte do programa que pode fazer referência à entidade declarada por meio de seu nome. Diz-se que tal entidade está “no escopo” dessa parte do programa. Esta seção apresenta várias questões importantes sobre o escopo.

As regras básicas do escopo são as seguintes:

1. O escopo de uma declaração de parâmetro é o corpo do método no qual a declaração aparece.
2. O escopo de uma declaração de variável local vai do ponto em que a declaração aparece até o final desse bloco.
3. O escopo de uma declaração de variável local que aparece na seção de inicialização do cabeçalho de uma instrução `for` é o corpo da instrução `for` e as outras expressões que estão no cabeçalho.
4. O escopo de um método ou campo é o corpo inteiro da classe. Isso permite que os métodos não estáticos de uma classe utilizem os campos e outros métodos da classe.

Qualquer bloco pode conter declarações de variável. Se uma variável local ou parâmetro em um método tem o mesmo nome de um campo da classe, o campo fica “oculto” até que termine a execução do bloco – isso é chamado de **sombreamento**. No Apêndice F, discutimos como acessar campos sombreados.



### Dica de prevenção de erro D.2

*Use nomes diferentes para campos e variáveis locais para ajudar a evitar erros de lógica sutis que ocorrem quando um método é chamado e uma variável local do método sombrea um campo na classe.*

A Figura D.6 demonstra as questões do escopo em campos e variáveis locais. A linha 7 declara e inicializa o campo `x` com 1. Esse campo é sombreado (oculto) em qualquer bloco (ou método) que declare uma variável local chamada `x`. O método `main` (linhas 11 a 23) declara uma variável local `x` (linha 13) e a inicializa com 5. O valor dessa variável local é produzido na saída para mostrar que o campo `x` (cujo valor é 1) é sombreado em `main`. O programa declara outros dois métodos – `useLocalVariable` (linhas 26 a 35) e `useField` (linhas 38 a 45) – que não recebem argumentos e não retornam resultados. O método `main` chama cada um desses métodos duas vezes (linhas 17 a 20). O método `useLocalVariable` declara a variável local `x` (linha 28). Quando `useLocalVariable` é chamado pela primeira vez (linha 17), cria uma variável local `x` e a inicializa com 25 (linha 28), gera o valor de `x` na saída (linhas 30 e 31), incrementa `x` (linha 32) e gera o valor de `x` na saída novamente (linhas 33 e 34). Quando `useLocalVariable` é chamado uma segunda vez (linha 19), recria a variável local `x` e a reinicializa com 25; portanto, a saída de cada chamada de `useLocalVariable` é idêntica.

```
1 // Fig. D.6: Scope.java
2 // A classe Scope demonstra os escopos de campo e variável local.
3
4 public class Scope
5 {
```

**Figura D.6** A classe `Scope` demonstra os escopos de campo e variável local. (continua)

```

6 // campo acessível a todos os métodos dessa classe
7 private static int x = 1;
8
9 // o método main cria e inicializa a variável local x
10 // e chama os métodos useLocalVariable e useField
11 public static void main(String[] args)
12 {
13 int x = 5; // a variável local x do método sombreia o campo x
14
15 System.out.printf("local x in main is %d\n", x);
16
17 useLocalVariable(); // useLocalVariable tem x local
18 useField(); // useField usa o campo x da classe Scope
19 useLocalVariable(); // useLocalVariable reinicializa x local
20 useField(); // o campo x da classe Scope mantém seu valor
21
22 System.out.printf("\nlocal x in main is %d\n", x);
23 } // fim de main
24
25 // cria e inicializa a variável local x durante cada chamada
26 public static void useLocalVariable()
27 {
28 int x = 25; // inicializada a cada vez que useLocalVariable é chamado
29
30 System.out.printf(
31 "\nlocal x on entering method useLocalVariable is %d\n", x);
32 ++x; // modifica a variável local x desse método
33 System.out.printf(
34 "local x before exiting method useLocalVariable is %d\n", x);
35 } // fim do método useLocalVariable
36
37 // modifica o campo x da classe Scope durante cada chamada
38 public static void useField()
39 {
40 System.out.printf(
41 "\nfield x on entering method useField is %d\n", x);
42 x *= 10; // modifica o campo x da classe Scope
43 System.out.printf(
44 "field x before exiting method useField is %d\n", x);
45 } // fim do método useField
46 } // fim da classe Scope

```

```

local x in main is 5

local x on entering method useLocalVariable is 25
local x before exiting method useLocalVariable is 26

field x on entering method useField is 1
field x before exiting method useField is 10

local x on entering method useLocalVariable is 25
local x before exiting method useLocalVariable is 26

field x on entering method useField is 10
field x before exiting method useField is 100

local x in main is 5

```

**Figura D.6** A classe Scope demonstra os escopos de campo e variável local.

O método `useField` não declara uma variável local. Portanto, quando ele faz referência a `x`, é usado o campo `x` (linha 7) da classe. Quando o método `useField` é chamado

pela primeira vez (linha 18), produz na saída o valor (1) do campo `x` (linhas 40 e 41), multiplica o campo `x` por 10 (linha 42) e novamente gera na saída o valor (1) do campo `x` (linhas 43 e 44) antes de retornar. Na próxima vez que o método `useField` é chamado (linha 20), o campo tem seu valor modificado (10), de modo que o método gera na saída 10 e depois 100. Por fim, no método `main`, o programa novamente gera na saída o valor da variável local `x` (linha 22) para mostrar que nenhuma das chamadas de método modificou a variável local `x` de `main`, pois todos os métodos fizeram referência às variáveis chamadas `x` em outros escopos.

## D.12 Sobrecarga de métodos

Métodos de mesmo nome podem ser declarados na mesma classe, desde que tenham conjuntos de parâmetros diferentes (determinados pelo número, tipo e ordem dos parâmetros) – isso se chama **sobrecarga de método**. Quando um método sobrecarregado é chamado, o compilador seleciona o método apropriado examinando o número, o tipo e a ordem dos argumentos na chamada. A sobrecarga de método é comumente usada para criar vários métodos com o *mesmo* nome e que executam a *mesma* tarefa ou tarefas *semelhantes*, mas com tipos ou números diferentes de argumentos. Por exemplo, os métodos `abs`, `min` e `max` de `Math` (resumidos na seção D.3) são sobrecarregados com quatro versões cada um:

1. Uma com dois parâmetros `double`.
2. Uma com dois parâmetros `float`.
3. Uma com dois parâmetros `int`.
4. Uma com dois parâmetros `long`.

Nosso próximo exemplo demonstra a declaração e a chamada de métodos sobrecarregados. Demonstramos os construtores sobrecarregados no Apêndice F.

### Declarando métodos sobrecarregados

A classe `MethodOverload` (Fig. D.7) inclui duas versões sobrecarregadas do método `square` – uma que calcula o quadrado de um valor `int` (e retorna um valor `int`) e outra que calcula o quadrado de um valor `double` (e retorna um valor `double`). Embora esses métodos tenham o mesmo nome e listas de parâmetros e corpos semelhantes, considere-os simplesmente como métodos *diferentes*. Talvez ajude a pensar nos nomes dos métodos como “quadrado de `int`” e “quadrado de `double`,” respectivamente.

```

1 // Fig. D.7: MethodOverload.java
2 // Declarações de métodos sobrecarregados.
3
4 public class MethodOverload
5 {
6 // testa os métodos square sobrecarregados
7 public static void main(String[] args)
8 {
9 System.out.printf("Square of integer 7 is %d\n", square(7));
10 System.out.printf("Square of double 7.5 is %f\n", square(7.5));
11 } // fim de

```

**Figura D.7** Declarações de métodos sobrecarregados. (*continua*)



```

12
13 // método square com argumento int
14 public static int square(int intValue)
15 {
16 System.out.printf("\nCalled square with int argument: %d\n",
17 intValue);
18 return intValue * intValue;
19 } // fim do método square com argumento int
20
21 // método square com argumento double
22 public static double square(double doubleValue)
23 {
24 System.out.printf("\nCalled square with double argument: %f\n",
25 doubleValue);
26 return doubleValue * doubleValue;
27 } // fim do método square com argumento double
28 } // fim da classe MethodOverload

```

```

Called square with int argument: 7
Square of integer 7 is 49

Called square with double argument: 7.500000
Square of double 7.5 is 56.250000

```

**Figura D.7** Declarações de métodos sobrecarregados.

A linha 9 chama o método `square` com o argumento 7. Os valores inteiros literais são tratados como tipo `int`, de modo que a chamada de método na linha 9 ativa a versão de `square` das linhas 14 a 19 que especifica um parâmetro `int`. Da mesma forma, a linha 10 chama o método `square` com o argumento 7.5. Os valores literais de ponto flutuante são tratados como tipo `double`, de modo que a chamada de método na linha 10 ativa a versão de `square` das linhas 22 a 27 que especifica um parâmetro `double`. Cada método primeiro gera na saída uma linha de texto para provar que o método correto foi chamado em cada caso. Os valores das linhas 10 e 24 são exibidos com o especificador de formato `%f`. Não especificamos precisão em nenhum dos casos. Por padrão, os valores de ponto flutuante são exibidos com seis dígitos de precisão, caso a precisão não seja definida no especificador de formato.

### ***Distinguindo entre métodos sobrecarregados***

O compilador distingue os métodos sobrecarregados pela sua **assinatura** – uma combinação do nome do método e do número, tipo e ordem de seus parâmetros. Se o compilador enxergasse somente os nomes de método durante a compilação, o código da Fig. D.7 seria ambíguo – o compilador não saberia como distinguir entre os dois métodos `square` (linhas 14 a 19 e 22 a 27). Internamente, o compilador usa nomes de método mais longos, que incluem o nome original do método, o tipo de cada parâmetro e a ordem exata dos parâmetros, para determinar se os métodos de uma classe são únicos nessa classe.

Por exemplo, na Fig. D.7, o compilador poderia usar o nome lógico “square de int” para o método `square` que especifica um parâmetro `int` e “square de double” para o método `square` que especifica um parâmetro `double` (os nomes reais utilizados pelo compilador são mais confusos). Se a declaração de `method1` começa como

```
void method1(int a, float b)
```

então, o compilador pode usar o nome lógico “method1 de int e float”. Se os parâmetros são especificados como

```
void method1(float a, int b)
```

então, o compilador pode usar o nome lógico “method1 de float e int”. A *ordem* dos tipos de parâmetro é importante – o compilador considera os dois cabeçalhos de method1 anteriores como distintos.

### **Tipos de retorno de métodos sobrecarregados**

Ao discutirmos os nomes lógicos dos métodos utilizados pelo compilador, não mencionamos os tipos de retorno dos métodos. *As chamadas de método não podem ser distinguidas pelo tipo de retorno.* Se você tivesse métodos sobrecarregados que diferissem apenas por seus tipos de retorno e se você chamasse um deles em uma instrução independente, como em:

```
square(2);
```

o compilador *não* poderia determinar a versão do método a ser chamada, pois o valor de retorno é ignorado. Quando dois métodos têm a mesma assinatura e tipos de retorno diferentes, o compilador gera uma mensagem de erro indicando que o método já está definido na classe. Os métodos sobrecarregados *podem* ter tipos de retorno diferentes, caso tenham listas de parâmetros diferentes. Além disso, os métodos sobrecarregados *não* precisam ter o mesmo número de parâmetros.



#### **Erro de programação comum D.5**

*Declarar métodos sobrecarregados com listas de parâmetros idênticas é um erro de compilação, independentemente de os tipos de retorno serem diferentes.*

## **D.13 Para finalizar**

Neste apêndice, você aprendeu mais sobre declarações de método. Aprendeu também a diferença entre métodos não estáticos e estáticos e como chamar métodos estáticos precedendo o nome do método com o nome da classe na qual aparece o método e o separador ponto (.). Você aprendeu a usar os operadores + e += para fazer concatenações de string. Discutimos como a pilha de chamada de métodos e os registros de ativação monitoram os métodos que foram chamados e para onde cada método deve retornar ao concluir sua tarefa. Discutimos também as regras de promoção da linguagem Java para fazer a conversão implícita entre os tipos primitivos e como fazer conversões explícitas com os operadores de conversão. Em seguida, você aprendeu sobre alguns dos pacotes mais comumente usados da API Java.

Você viu como declara constantes nomeadas utilizando tipos enum e variáveis private static final. Usou a classe Random para gerar números aleatórios para simulações. Aprendeu também sobre o escopo de campos e variáveis locais em uma classe. Por último, aprendeu que vários métodos de uma classe podem ser sobrecarregados, fornecendo-se métodos com o mesmo nome e assinaturas diferentes. Esses métodos podem ser usados para executar a mesma tarefa ou tarefas semelhantes, com tipos ou números diferentes de parâmetros.

No Apêndice E, você vai aprender a manter listas e tabelas de dados em arrays. Vai ver uma implementação mais elegante do aplicativo que lança um dado 6.000.000 de vezes e duas versões melhoradas de nosso estudo de caso GradeBook, que analisamos nos Apêndices B e C. Você também vai aprender a acessar os argumentos de linha de comando de um aplicativo, que são passados para o método main quando o aplicativo começa a ser executado.

## Exercícios de revisão

**D.1** Preencha os espaços em branco em cada um dos seguintes enunciados:

- a) Um método é ativado com um(a) \_\_\_\_\_.
- b) Uma variável conhecida somente dentro do método em que é declarada é chamada de \_\_\_\_\_.
- c) A instrução \_\_\_\_\_ em um método chamado pode ser usada para passar o valor de uma expressão de volta para o método chamador.
- d) A palavra-chave \_\_\_\_\_ indica que um método não retorna um valor.
- e) Dados só podem ser adicionados ou removidos do \_\_\_\_\_ de uma pilha.
- f) As pilhas são conhecidas como estruturas de dados \_\_\_\_\_; o último item colocado (inserido) na pilha é o primeiro item retirado (removido) da pilha.
- g) As três maneiras de retornar o controle de um método chamado para um chamador são \_\_\_\_\_, \_\_\_\_\_ e \_\_\_\_\_.
- h) Um objeto da classe \_\_\_\_\_ produz números aleatórios.
- i) A pilha de execução do programa contém a memória para as variáveis locais em cada chamada de um método durante a execução de um programa. Esses dados, armazenados como parte da pilha de execução do programa, são conhecidos como \_\_\_\_\_ ou \_\_\_\_\_ da chamada de método.
- j) Se existem mais chamadas de método do que podem ser armazenadas na pilha de execução do programa, ocorre um erro conhecido como \_\_\_\_\_.
- k) O \_\_\_\_\_ de uma declaração é a parte de um programa que pode fazer referência à entidade declarada por meio de seu nome.
- l) É possível ter vários métodos com o mesmo nome, cada um operando em diferentes tipos ou números de argumentos. Esse recurso é chamado de \_\_\_\_\_ de método.
- m) A pilha de execução do programa também é referida como pilha de \_\_\_\_\_.

**D.2** Para a classe Craps da Fig. D.5, indique o escopo de cada uma das seguintes entidades:

- a) a variável `randomNumbers`.
- b) a variável `die1`.
- c) o método `rollDice`.
- d) o método `main`.
- e) a variável `sumOfDice`.

**D.3** Escreva um aplicativo que teste se os exemplos de chamadas de método da classe `Math` mostrados na Fig. D.1 produzem realmente os resultados indicados.

**D.4** Forneça o cabeçalho de cada um dos seguintes métodos:

- a) Método `hypotenuse`, que recebe dois argumentos de ponto flutuante e precisão dupla, `side1` e `side2`, e retorna um resultado de ponto flutuante e precisão dupla.
- b) Método `smallest`, que recebe três valores inteiros `x`, `y` e `z` e retorna um valor inteiro.
- c) Método `instructions`, que não recebe argumento e não retorna valor. [*Obs.*: tais métodos são comumente usados para mostrar instruções para um usuário.]
- d) Método `intToFloat`, que recebe um argumento `number` inteiro e retorna um valor `float`.

**D.5** Encontre o erro em cada um dos seguintes trechos de programa. Explique como corrigi-lo.

```
a) void g()
{
 System.out.println("Inside method g");

 void h()
 {
 System.out.println("Inside method h");
 }
}

b) int sum(int x, int y)
{
 int result;
 result = x + y;
}

c) void f(float a);
{
 float a;
 System.out.println(a);
}
```

**D.6** Escreva um aplicativo Java completo que solicite ao usuário o raio de tipo `double` de uma esfera e chame o método `sphereVolume` para calcular e exibir o volume da esfera. Use a seguinte instrução para calcular o volume:

```
double volume = (4.0 / 3.0) * Math.PI * Math.pow(radius, 3)
```

## Respostas dos exercícios de revisão

**D.1** a) chamada de método. b) variável local. c) `return`. d) `void`. e) topo. f) LIFO (último a entrar, primeiro a sair – *last-in, first-out*). g) `return`; ou *expressão* `return`; ou encontrar a chave de fechamento de um método. h) `Random`. i) registro de ativação, `stack frame`. j) estouro da pilha. k) `escopo`. l) *sobrecarga*. m) chamada de métodos.

**D.2** a) o corpo da classe. b) o bloco que define o corpo do método `rollDice`. c) o corpo da classe. d) o corpo da classe. e) o bloco que define o corpo do método `main`.

**D.3** A solução a seguir demonstra os métodos da classe `Math` da Fig. D.1:

```
1 // Exercício D.3: MathTest.java
2 // Teste dos métodos da classe Math.
3
4 public class MathTest
5 {
6 public static void main(String[] args)
7 {
8 System.out.printf("Math.abs(23.7) = %f\n", Math.abs(23.7));
9 System.out.printf("Math.abs(0.0) = %f\n", Math.abs(0.0));
10 System.out.printf("Math.abs(-23.7) = %f\n", Math.abs(-23.7));
11 System.out.printf("Math.ceil(9.2) = %f\n", Math.ceil(9.2));
12 System.out.printf("Math.ceil(-9.8) = %f\n", Math.ceil(-9.8));
13 System.out.printf("Math.cos(0.0) = %f\n", Math.cos(0.0));
14 System.out.printf("Math.exp(1.0) = %f\n", Math.exp(1.0));
15 System.out.printf("Math.exp(2.0) = %f\n", Math.exp(2.0));
16 System.out.printf("Math.floor(9.2) = %f\n", Math.floor(9.2));
17 System.out.printf("Math.floor(-9.8) = %f\n",
18 Math.floor(-9.8));
19 System.out.printf("Math.log(Math.E) = %f\n",
20 Math.log(Math.E));
21 System.out.printf("Math.log(Math.E * Math.E) = %f\n",
22 Math.log(Math.E * Math.E));
23 System.out.printf("Math.max(2.3, 12.7) = %f\n",
24 Math.max(2.3, 12.7));
25 System.out.printf("Math.max(-2.3, -12.7) = %f\n",
```

(continua)

```

26 Math.max(-2.3, -12.7));
27 System.out.printf("Math.min(2.3, 12.7) = %f\n",
28 Math.min(2.3, 12.7));
29 System.out.printf("Math.min(-2.3, -12.7) = %f\n",
30 Math.min(-2.3, -12.7));
31 System.out.printf("Math.pow(2.0, 7.0) = %f\n",
32 Math.pow(2.0, 7.0));
33 System.out.printf("Math.pow(9.0, 0.5) = %f\n",
34 Math.pow(9.0, 0.5));
35 System.out.printf("Math.sin(0.0) = %f\n", Math.sin(0.0));
36 System.out.printf("Math.sqrt(900.0) = %f\n",
37 Math.sqrt(900.0));
38 System.out.printf("Math.tan(0.0) = %f\n", Math.tan(0.0));
39 } // fim de main
40 } // fim da classe MathTest

```

```

Math.abs(23.7) = 23.700000
Math.abs(0.0) = 0.000000
Math.abs(-23.7) = 23.700000
Math.ceil(9.2) = 10.000000
Math.ceil(-9.8) = -9.000000
Math.cos(0.0) = 1.000000
Math.exp(1.0) = 2.718282
Math.exp(2.0) = 7.389056
Math.floor(9.2) = 9.000000
Math.floor(-9.8) = -10.000000
Math.log(Math.E) = 1.000000
Math.log(Math.E * Math.E) = 2.000000
Math.max(2.3, 12.7) = 12.700000
Math.max(-2.3, -12.7) = -2.300000
Math.min(2.3, 12.7) = 2.300000
Math.min(-2.3, -12.7) = -12.700000
Math.pow(2.0, 7.0) = 128.000000
Math.pow(9.0, 0.5) = 3.000000
Math.sin(0.0) = 0.000000
Math.sqrt(900.0) = 30.000000
Math.tan(0.0) = 0.000000

```

- D.4**
- a) `double` hypotenuse( `double` side1, `double` side2 )
  - b) `int` smallest( `int` x, `int` y, `int` z )
  - c) `void` instructions()
  - d) `float` intToFloat( `int` number )

- D.5**
- a) Erro: o método `h` está declarado dentro do método `g`.  
Correção: mover a declaração de `h` para fora da declaração de `g`.
  - b) Erro: o método deveria retornar um valor inteiro, mas não retorna.  
Correção: excluir a variável `result` e colocar a instrução

```
return x + y;
```

no método, ou adicionar a seguinte instrução no final do corpo do método:

```
return result;
```

- c) Erro: o ponto e vírgula após o parêntese de abertura da lista de parâmetros está incorreto e o parâmetro `a` não deve ser redeclarado no método.  
Correção: excluir o ponto e vírgula após o parêntese de abertura da lista de parâmetros e excluir a declaração `float a;`.

**D.6** A solução a seguir calcula o volume de uma esfera usando o raio digitado pelo usuário:

```

1 // Exercício D.6: Sphere.java
2 // Calcula o volume de uma esfera.
3 import java.util.Scanner;
4
5 public class Sphere
6 {
7 // obtém o raio do usuário e exibe o volume da esfera
8 public static void main(String[] args)
9 {
10 Scanner input = new Scanner(System.in);
11 System.out.print("Enter radius of sphere: ");
12 double radius = input.nextDouble();
13 System.out.printf("Volume is %f\n", sphereVolume(radius));
14 } // fim do método determineSphereVolume
15
16 // calcula e retorna o volume da esfera
17 public static double sphereVolume(double radius)
18 {
19 double volume = (4.0 / 3.0) * Math.PI * Math.pow(radius, 3);
20 return volume;
21 } // fim do método sphereVolume
22 } // fim da classe Sphere

```

```

Enter radius of sphere: 4
Volume is 268.082573

```

## Exercícios

**D.7** Qual é o valor de  $x$  após a execução das instruções a seguir?

- a)  $x = \text{Math.abs}( 7.5 );$
- b)  $x = \text{Math.floor}( 7.5 );$
- c)  $x = \text{Math.abs}( 0.0 );$
- d)  $x = \text{Math.ceil}( 0.0 );$
- e)  $x = \text{Math.abs}( -6.4 );$
- f)  $x = \text{Math.ceil}( -6.4 );$
- g)  $x = \text{Math.ceil}( -\text{Math.abs}( -8 + \text{Math.floor}( -5.5 ) ) );$

**D.8** (*Preços de estacionamento*) Um estacionamento cobra um valor mínimo de US\$2.00 por 3 horas. Há uma taxa adicional de US\$0.50 por hora para cada hora cheia ou fração que ultrapasse as 3 horas. O valor máximo cobrado por um período de 24 horas é de US\$10.00. Suponha que nenhum carro pode ficar estacionado por mais de 24 horas por vez. Escreva um aplicativo que calcule e exiba as taxas de estacionamento para cada cliente que utilizou uma vaga ontem. Você deve inserir as horas estacionadas por cada cliente. O programa deve mostrar a taxa para cada cliente, calcular e exibir o total parcial recebido ontem. Deve usar o método `calculateCharges` para determinar a taxa de cada cliente.

**D.9** (*Números arredondados*) `Math.floor` pode ser usado para arredondar valores para o inteiro mais próximo. Por exemplo,

```
y = Math.floor(x + 0.5);
```

arredondará o número  $x$  para o inteiro mais próximo e atribuirá o resultado a  $y$ . Escreva um aplicativo que leia valores `double` e utilize a instrução anterior para arredondar cada um dos números para o inteiro mais próximo. Para cada número processado, exiba o original e o arredondado.

- D.10 (Números arredondados)** Para arredondar números com casas decimais específicas, use uma instrução como

```
y = Math.floor(x * 10 + 0.5) / 10;
```

a qual arredonda  $x$  na posição dos décimos (isto é, a primeira posição à direita do ponto decimal), ou

```
y = Math.floor(x * 100 + 0.5) / 100;
```

a qual arredonda  $x$  na posição dos centésimos (isto é, a segunda posição à direita do ponto decimal). Escreva um aplicativo que defina quatro métodos para arredondar um número  $x$  de várias maneiras:

- `roundToInteger( number )`
- `roundToTenths( number )`
- `roundToHundredths( number )`
- `roundToThousandths( number )`

Para cada valor lido, seu programa deve exibir o valor original, o número arredondado para o inteiro mais próximo, o número arredondado para o décimo mais próximo, o número arredondado para o centésimo mais próximo e o número arredondado para o milésimo mais próximo.

- D.11** Responda cada uma das seguintes perguntas:

- O que significa escolher números “aleatoriamente”?
- Por que o método `nextInt` da classe `Random` é útil para simular jogos de azar?
- Por que frequentemente é necessário mudar a escala ou deslocar os valores produzidos por um objeto `Random`?
- Por que a simulação computadorizada de situações reais é uma técnica útil?

- D.12** Escreva instruções que atribuam valores inteiros aleatórios à variável  $n$  nos seguintes intervalos:

- $1 \leq n \leq 2$ .
- $1 \leq n \leq 100$ .
- $0 \leq n \leq 9$ .
- $1000 \leq n \leq 1112$ .
- $-1 \leq n \leq 1$ .
- $-3 \leq n \leq 11$ .

- D.13** Escreva instruções que exibam um número aleatório de cada um dos seguintes conjuntos:

- 2, 4, 6, 8, 10.
- 3, 5, 7, 9, 11.
- 6, 10, 14, 18, 22.

- D.14 (Exponenciação)** Escreva um método `integerPower(base, exponent)` que retorne o valor de  $base^{exponent}$

Por exemplo, `integerPower(3, 4)` calcula  $3^4$  (ou  $3 * 3 * 3 * 3$ ). Suponha que `exponent` é um valor inteiro positivo diferente de zero e que `base` é um valor inteiro. Use uma instrução `for` ou `while` para controlar o cálculo. Não use um método da classe `Math`. Incorpore esse método a um aplicativo que leia valores inteiros para `base` e `exponent` e efetue o cálculo com o método `integerPower`.

- D.15 (Múltiplos)** Escreva um método `isMultiple` que determine, para dois valores inteiros, se o segundo é múltiplo do primeiro. O método deve receber dois argumentos inteiros e retornar `true` se o segundo for múltiplo do primeiro e `false`, caso contrário. [Dica: use o operador

resto.] Incorpore esse método a um aplicativo que insira uma série de pares de valores inteiros (um par por vez) e determine se o segundo valor de cada par é múltiplo do primeiro.

**D.16** (*Par ou ímpar*) Escreva um método `isEven` que utilize o operador resto (%) para determinar se um valor inteiro é par. O método deve receber um argumento inteiro e retornar `true` se o valor inteiro for par e `false`, caso contrário. Incorpore esse método a um aplicativo que insira uma sequência de valores inteiros (um por vez) e determine se cada um é par ou ímpar.

**D.17** (*Área do círculo*) Escreva um aplicativo que solicite ao usuário o raio de um círculo e utilize um método chamado `circleArea` para calcular a área do círculo.

**D.18** (*Conversões de temperatura*) Implemente os seguintes métodos inteiros:

a) O método `celsius` retorna o equivalente em Celsius de uma temperatura em Fahrenheit, usando o cálculo

$$\text{celsius} = 5.0 / 9.0 * (\text{fahrenheit} - 32);$$

b) O método `fahrenheit` retorna o equivalente em Fahrenheit de uma temperatura em Celsius, usando o cálculo

$$\text{fahrenheit} = 9.0 / 5.0 * \text{celsius} + 32;$$

c) Use os métodos das partes (a) e (b) para escrever um aplicativo que permita ao usuário digitar uma temperatura em Fahrenheit e exibir a equivalente em Celsius, ou digitar uma temperatura em Celsius e exibir a equivalente em Fahrenheit.

**D.19** (*Encontre o mínimo*) Escreva um método `minimum3` que retorne o menor de três números de ponto flutuante. Use o método `Math.min` para implementar `minimum3`. Incorpore o método a um aplicativo que leia três valores do usuário, determine o menor valor e exiba o resultado.

**D.20** (*Máximo divisor comum*) O *máximo divisor comum* (MDC) de dois valores inteiros é o maior inteiro que divide igualmente cada um dos dois números. Escreva um método que retorne o máximo divisor comum de dois valores inteiros. [Dica: talvez você queira usar o algoritmo de Euclides. Informações sobre ele podem ser encontradas em [en.wikipedia.org/wiki/Euclidean\\_algorithm](http://en.wikipedia.org/wiki/Euclidean_algorithm).] Incorpore o método a um aplicativo que leia dois valores do usuário e exiba o resultado.

**D.21** (*Pontos de qualificação*) Escreva um método `qualityPoints` que insira a média de um aluno e retorne 4 se estiver entre 90 e 100, 3 se estiver entre 80 e 89, 2 se estiver entre 70 e 79, 1 se estiver entre 60 e 69 e 0 se for menor que 60. Incorpore o método a um aplicativo que leia um valor do usuário e exiba o resultado.

**D.22** (*Lançamento de moeda*) Escreva um aplicativo que simule o lançamento de uma moeda. Faça o programa lançar uma moeda sempre que o usuário escolher a opção de menu “Lançar Moeda”. Conte o número de vezes que cada lado da moeda aparece. Exiba os resultados. O programa deve chamar um método separado `flip` que não recebe argumentos e que retorna um valor de `enum Coin` (HEADS e TAILS). [Obs.: se o programa simula realisticamente o lançamento de uma moeda, cada lado da moeda deve aparecer aproximadamente metade das vezes.]

**D.23** (*Adivinhe o número*) Escreva um aplicativo para “adivinhar o número” como segue: seu programa escolhe o número a ser adivinhado selecionando um valor inteiro aleatório no intervalo de 1 a 1000. O aplicativo exibe o prompt `Adivinhe um número entre 1 e 1000`. O jogador insere o primeiro palpite. Se o palpite do jogador estiver incorreto, seu programa deve exibir “Muito alto. Tente outra vez.” ou “Muito baixo. Tente outra vez.” para ajudá-lo a “fechar o cerco” na resposta correta. O programa deve pedir o próximo palpite ao usuário. Quando o usuário digitar a resposta correta, exiba “Parabéns. Você adivinhou o número!”, e permita que ele escolha se vai jogar de novo. A técnica de adivinhação empregada neste problema é semelhante a uma busca binária.



**D.24 (Modificação do jogo craps)** Modifique o programa de *craps* da Fig. D.5 para permitir apostas. Inicialize a variável `bankBalance` com 1000 dólares. Peça ao jogador para digitar uma aposta (`wager`). Verifique se `wager` é menor ou igual a `bankBalance` e, se não for, peça ao usuário para que digite a aposta novamente até que seja inserido um valor válido. Então, execute um jogo de *craps*. Se o jogador ganhar, aumente `bankBalance` por `wager` e exiba o novo valor de `bankBalance`. Se o jogador perder, diminua `bankBalance` por `wager`, exiba o novo valor de `bankBalance`, verifique se `bankBalance` se tornou zero e, em caso positivo, exiba a mensagem "Desculpe. Você faliu!". À medida que o jogo avançar, exiba várias mensagens para criar algum "palavreado", como "Oh, você vai ficar sem dinheiro, hein?" ou "Ah, vamos lá, arrisque!" ou "Você é muito bom. Agora é hora de usar suas fichas!". Implemente o "palavreado" como um método separado que escolha aleatoriamente a string a ser exibida.

**D.25 (Instrução Assistida por Computador)** O uso de computadores na educação é conhecido como *CAI* (*Instrução Assistida por Computador – do inglês Computer-Assisted Instruction*). Escreva um programa que ajude um aluno do ensino fundamental a aprender multiplicação. Use um objeto `Random` para produzir dois valores inteiros positivos com um algarismo. O programa deve então fazer uma pergunta ao usuário, como

Quanto é 6 vezes 7?

O aluno digita a resposta. Em seguida, o programa verifica a resposta do aluno. Se estiver certa, exibe a mensagem "Muito bom!" e faz outra pergunta sobre multiplicação. Se a resposta for errada, exibe a mensagem "Não. Tente outra vez." e permite que o aluno tente a mesma pergunta repetidamente, até finalmente acertar. Um método separado deve ser usado para gerar cada nova pergunta. Esse método deve ser chamado uma vez quando o aplicativo começar a ser executado e sempre que o usuário responder à questão corretamente.

**D.26 (Instrução Assistida por Computador: reduzindo a fadiga do aluno)** Um problema nos ambientes de CAI é a fadiga do aluno. Ela pode ser reduzida variando-se as respostas do computador para manter a atenção do aluno. Modifique o programa do Exercício D.25 de modo que vários comentários sejam exibidos para cada resposta, como segue:

Possíveis reações para uma resposta correta:

Muito bem!  
Excelente!  
Bom trabalho!  
Continue assim!

Possíveis reações para uma resposta incorreta:

Não. Tente outra vez.  
Errado. Tente de novo.  
Não desista!  
Não. Continue tentando.

Use geração de números aleatórios para escolher um número de 1 a 4, que será usado para selecionar uma das quatro reações apropriadas para cada resposta correta ou incorreta. Use uma instrução `switch` para emitir as reações.

**D.27 (Instrução Assistida por Computador: variando os tipos de problemas)** Modifique o programa anterior para permitir que o usuário escolha um tipo de problema de aritmética para estudar. A opção 1 significa somente problemas de adição, 2 significa somente problemas de subtração, 3 significa somente problemas de multiplicação, 4 significa somente problemas de divisão e 5 significa uma mistura aleatória de todos esses tipos.

# Arrays e ArrayLists

# E

## Objetivos

Neste capítulo, você vai:

- Aprender o que são arrays.
- Usar arrays para armazenar e recuperar dados em listas e tabelas de valores.
- Declarar arrays, inicializar arrays e fazer referência a elementos individuais de arrays.
- Iterar por arrays com a instrução `for` melhorada.
- Passar arrays para métodos.
- Declarar e manipular arrays multidimensionais.
- Realizar manipulações comuns de array com os métodos da classe `Arrays`.
- Usar a classe `ArrayList` para manipular uma estrutura de dados do tipo array redimensionável dinamicamente.



- |                                                                                 |                                                                                     |
|---------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <b>E.1</b> Introdução                                                           | <b>E.8</b> Estudo de caso: classe GradeBook usando um array para armazenar as notas |
| <b>E.2</b> Arrays                                                               | <b>E.9</b> Arrays multidimensionais                                                 |
| <b>E.3</b> Declaração e criação de arrays                                       | <b>E.10</b> Estudo de caso: classe GradeBook usando um array bidimensional          |
| <b>E.4</b> Exemplos de uso de arrays                                            | <b>E.11</b> Classe Arrays                                                           |
| <b>E.5</b> Estudo de caso: simulação de embaralhamento e distribuição de cartas | <b>E.12</b> Introdução às coleções e à classe ArrayList                             |
| <b>E.6</b> Instrução for melhorada                                              | <b>E.13</b> Para finalizar                                                          |
| <b>E.7</b> Passagem de arrays para métodos                                      |                                                                                     |

*Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios*

## E.1 Introdução

Este apêndice apresenta as **estruturas de dados** – coleções de itens de dados relacionados. Os **arrays** são estruturas de dados compostas de itens de dados relacionados do mesmo tipo. Eles tornam conveniente processar grupos de valores relacionados. Uma vez criados, os arrays permanecem com o mesmo comprimento, embora uma variável de array possa ter uma nova atribuição, de modo a fazer referência a um novo array de comprimento diferente.

Embora sejam comumente usados, os arrays têm capacidades limitadas. Por exemplo, você deve especificar o tamanho de um array e, se quiser modificá-lo no momento da execução, deve fazer isso manualmente, criando um novo array. No final deste apêndice, apresentamos uma das estruturas de dados predefinidas da linguagem Java, das classes de coleção da API Java. Essas estruturas oferecem mais recursos do que os arrays tradicionais. Nos concentramos na coleção ArrayList. ArrayLists são semelhantes aos arrays, mas oferecem funcionalidade adicional, como o **redimensionamento dinâmico** – seu tamanho aumenta automaticamente em tempo de execução para acomodar mais elementos.

## E.2 Arrays

Um array é um grupo de variáveis (chamadas de **elementos** ou **componentes**) contendo valores, todos do mesmo tipo. Os arrays são *objetos*; portanto, são considerados tipos de referência. Conforme você vai ver em breve, o que normalmente consideramos um array é na verdade uma referência para um objeto array na memória. Os *elementos* de um array podem ser tipos primitivos ou tipos de referência (incluindo arrays, como veremos na seção E.9). Para fazer referência a um elemento em particular de um array, especificamos o nome da referência ao array e o *número da posição* do elemento no array. O número da posição do elemento é denominado **índice** ou **subscrito** do elemento.

A Figura E.1 mostra uma representação lógica de um array inteiro chamado *c*. Esse array contém 12 elementos. Um programa refere-se a qualquer um desses elementos com uma **expressão de acesso ao array** que inclui o nome do array, seguido do índice do elemento em particular entre **colchetes** (`[]`). O primeiro elemento de todo array tem o **índice zero** e às vezes é chamado de **elemento zero**. Assim, os elementos do array *c* são *c*[0], *c*[1], *c*[2] e assim por diante. O índice mais alto no array *c* é 11, que é 1 a menos que 12 – o número de elementos no array. Os nomes de array seguem as mesmas convenções de outros nomes de variável.

|                                               |         |      |
|-----------------------------------------------|---------|------|
| Nome do array (c) →                           | c[ 0 ]  | -45  |
|                                               | c[ 1 ]  | 6    |
|                                               | c[ 2 ]  | 0    |
|                                               | c[ 3 ]  | 72   |
|                                               | c[ 4 ]  | 1543 |
|                                               | c[ 5 ]  | -89  |
|                                               | c[ 6 ]  | 0    |
|                                               | c[ 7 ]  | 62   |
|                                               | c[ 8 ]  | -3   |
|                                               | c[ 9 ]  | 1    |
|                                               | c[ 10 ] | 6453 |
| Índice (ou subcrito) do elemento no array c → | c[ 11 ] | 78   |

**Figura E.1** Um array de 12 elementos.

O índice deve ser um valor inteiro não negativo. Um programa pode usar uma expressão como índice. Por exemplo, se supomos que a variável *a* é 5 e a variável *b* é 6, então a instrução

```
c[a + b] += 2;
```

soma 2 ao elemento do array *c*[11]. O nome de um array indexado é uma expressão de acesso ao array, a qual pode ser usada no lado esquerdo de uma atribuição para inserir um novo valor em um elemento do array.



#### Erro de programação comum E.1

*Um índice deve ser um valor int ou um valor de um tipo que possa ser promovido a int – ou seja, byte, short ou char, mas não long; caso contrário, ocorrerá um erro de compilação.*

Vamos examinar o array *c* da Fig. E.1 mais detidamente. O **nome** do array é *c*. Todo objeto array conhece seu próprio comprimento e o armazena em uma **variável de instância** *length*. A expressão *c.length* acessa o campo *length* do array *c* para determinar o comprimento do array. Mesmo a variável de instância *length* de um array sendo *public*, ela não pode ser alterada, pois é uma variável *final*. Os 12 elementos desse array são referidos como *c*[0], *c*[1], *c*[2], ..., *c*[11]. O valor de *c*[0] é -45, o valor de *c*[1] é 6, o valor de *c*[2] é 0, o valor de *c*[7] é 62 e o valor de *c*[11] é 78. Para calcular a soma dos valores contidos nos três primeiros elementos do array *c* e armazenar o resultado na variável *sum*, escreveríamos

```
sum = c[0] + c[1] + c[2];
```

Para dividir o valor de *c*[6] por 2 e atribuir o resultado à variável *x*, escreveríamos

```
x = c[6] / 2;
```

## E.3 Declaração e criação de arrays

Os objetos array ocupam espaço na memória. Assim como outros objetos, os arrays são criados com a palavra-chave *new*. Para criar um objeto array, especifique o tipo dos

elementos do array e o número de elementos como parte de uma **expressão de criação de array** que utilize a palavra-chave `new`. Essa expressão retorna uma referência que pode ser armazenada em uma variável de array. A declaração e a expressão de criação de array a seguir criam um objeto array contendo 12 elementos `int` e armazenam a referência do array na variável de array `c`:

```
int[] c = new int[12];
```

Essa expressão pode ser usada para criar o array mostrado na Fig. E.1. Quando um array é criado, cada elemento dele recebe um valor padrão – zero para os elementos numéricos de tipo primitivo, `false` para elementos `boolean` e `null` para referências. Conforme você vai ver em breve, é possível fornecer valores de elemento iniciais não padrão ao criar um array. A criação do array da Fig. E.1 também pode ser realizada em duas etapas, como segue:

```
int[] c; // declara a variável de array
c = new int[12]; // cria o array; atribui à variável de array
```

Na declaração, os colchetes após o tipo indicam que `c` é uma variável que vai fazer referência a um array (isto é, a variável vai armazenar uma referência de array). Na instrução de atribuição, a variável de array `c` recebe a referência para um novo array de 12 elementos `int`.

Um programa pode criar vários arrays em uma única declaração. A declaração a seguir reserva 100 elementos para `b` e 27 elementos para `x`:

```
String[] b = new String[100], x = new String[27];
```

Quando o tipo do array e os colchetes são combinados no início da declaração, todos os identificadores na declaração são variáveis de array. Neste caso, as variáveis `b` e `x` se referem a arrays `String`. Por clareza, preferimos declarar apenas uma variável por declaração. A declaração anterior é equivalente a:

```
String[] b = new String[100]; // cria o array b
String[] x = new String[27]; // cria o array x
```

Quando somente uma variável é declarada em cada declaração, os colchetes podem ser colocados depois do tipo ou depois do nome da variável de array, como em:

```
String b[] = new String[100]; // cria o array b
String x[] = new String[27]; // cria o array x
```



### Erro de programação comum E.2

*Declarar várias variáveis de array em uma única declaração pode levar a erros sutis.*

*Considere a declaração `int[] a, b, c;`. Se `a`, `b` e `c` vão ser declarados como variáveis de array, então essa declaração está correta – a colocação de colchetes imediatamente após o tipo indica que todos os identificadores da declaração são variáveis de array. Contudo, se apenas `a` vai ser uma variável de array, e `b` e `c` vão ser variáveis `int` individuais, então essa declaração está incorreta – a declaração `int a[], b, c;` obterá o resultado desejado.*

Um programa pode declarar arrays de qualquer tipo. Todo elemento de um array de tipo primitivo contém um valor do tipo do elemento declarado do array. Do mesmo modo, em um array de tipo de referência, todo elemento é uma referência para um objeto do tipo do elemento declarado do array. Por exemplo, todo elemento de um array `int` é um valor `int`, e todo elemento de um array `String` é uma referência para um objeto `String`.

## E.4 Exemplos de uso de arrays

Esta seção apresenta vários exemplos que demonstram a declaração, a criação e a inicialização de arrays e a manipulação de seus elementos.

### Criando e inicializando um array

O aplicativo da Fig. E.2 usa a palavra-chave `new` para criar um array de 10 elementos `int`, os quais são inicialmente zero (o padrão para variáveis `int`). A linha 8 declara o array – uma referência capaz de se referir a um array de elementos `int`. A linha 10 cria o objeto array e atribui sua referência à variável `array`. A linha 12 gera os cabeçalhos de coluna na saída. A primeira coluna contém o índice (de 0 a 9) de cada elemento do array, e a segunda contém o valor padrão (0) de cada elemento.

```

1 // Fig. E.2: InitArray.java
2 // Inicializando os elementos de um array com valores padrão zero.
3
4 public class InitArray
5 {
6 public static void main(String[] args)
7 {
8 int[] array; // declara o array chamado array
9
10 array = new int[10]; // cria o objeto array
11
12 System.out.printf("%5s%8s\n", "Index", "Value"); // cabeçalhos de coluna
13
14 // gera na saída o valor de cada elemento de array
15 for (int counter = 0; counter < array.length; counter++)
16 System.out.printf("%5d%8d\n", counter, array[counter]);
17 } // fim de main
18 } // fim da classe InitArray

```

| Índice | Valor |
|--------|-------|
| 0      | 0     |
| 1      | 0     |
| 2      | 0     |
| 3      | 0     |
| 4      | 0     |
| 5      | 0     |
| 6      | 0     |
| 7      | 0     |
| 8      | 0     |
| 9      | 0     |

**Figura E.2** Inicializando os elementos de um array com valores padrão zero.

A instrução `for` nas linhas 15 e 16 gera o número do índice na saída (representado por `counter`) e o valor de cada elemento do array (representado por `array[counter]`). A variável de controle do loop `counter` é inicialmente 0 – os valores de índice começam em 0; portanto, usar **contagem baseada em zero** permite que o loop acesse cada elemento do array. A condição de continuação do loop `for` usa a expressão `array.length` (linha 15) para determinar o comprimento do array. Neste exemplo, o comprimento do array é 10, de modo que o loop continua a executar enquanto o valor da variável de controle `counter` é menor que 10. O valor de índice mais alto de um array de 10 elementos é 9; portanto, usar o operador menor que na condição de continuação do loop garante que o loop não tentará acessar um elemento *além* do final do array (isto é, durante a última

iteração do loop, counter é 9). Em breve, vamos ver o que o Java faz ao encontrar tal *índice fora do intervalo* no momento da execução.

### Usando um inicializador de array

Você pode criar um array e inicializar seus elementos com um **inicializador de array** – uma lista de expressões separadas por vírgulas (chamada de **lista de inicializadores**) entre chaves. Nesse caso, o comprimento do array é determinado pelo número de elementos na lista de inicializadores. Por exemplo,

```
int[] n = { 10, 20, 30, 40, 50 };
```

cria um array de cinco elementos com valores de índice de 0 a 4. O elemento `n[0]` é inicializado com 10, `n[1]` é inicializado com 20 e assim por diante. Quando o compilador encontra uma declaração de array que inclui uma lista de inicializadores, ele conta o número de inicializadores na lista para determinar o tamanho do array e, então, configura a operação `new` apropriada “nos bastidores”.

O aplicativo da Fig. E.3 inicializa um array inteiro com 10 valores (linha 9) e o exibe em formato tabular. O código para exibir os elementos do array (linhas 14 e 15) é idêntico ao da Fig. E.2 (linhas 15 e 16).

```
1 // Fig. E.3: InitArray.java
2 // Inicializando os elementos de um array com um inicializador de array.
3
4 public class InitArray
5 {
6 public static void main(String[] args)
7 {
8 // a lista de inicializadores especifica o valor de cada elemento
9 int[] array = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
10
11 System.out.printf("%s%8s\n", "Index", "Value"); // cabeçalhos de coluna
12
13 // gera o valor de cada elemento do array na saída
14 for (int counter = 0; counter < array.length; counter++)
15 System.out.printf("%5d%8d\n", counter, array[counter]);
16 } // fim de main
17 } // fim da classe InitArray
```

| Índice | Valor |
|--------|-------|
| 0      | 32    |
| 1      | 27    |
| 2      | 64    |
| 3      | 18    |
| 4      | 95    |
| 5      | 14    |
| 6      | 90    |
| 7      | 70    |
| 8      | 60    |
| 9      | 37    |

**Figura E.3** Inicializando os elementos de um array com um inicializador de array.

### Calculando os valores a serem armazenados em um array

O aplicativo da Fig. E.4 cria um array de 10 elementos e atribui a cada elemento um dos valores inteiros pares de 2 a 20 (2, 4, 6, ..., 20). Então, o aplicativo exibe o array em formato tabular. A instrução `for` nas linhas 12 e 13 calcula o valor de um elemento do array multiplicando o valor atual da variável de controle `counter` por 2 e somando 2.

```

1 // Fig. E.4: InitArray.java
2 // Calculando os valores a serem colocados nos elementos de um array.
3
4 public class InitArray
5 {
6 public static void main(String[] args)
7 {
8 final int ARRAY_LENGTH = 10; // declara a constante
9 int[] array = new int[ARRAY_LENGTH]; // cria o array
10
11 // calcula o valor de cada elemento do array
12 for (int counter = 0; counter < array.length; counter++)
13 array[counter] = 2 + 2 * counter;
14
15 System.out.printf("%s%s\n", "Index", "Value"); // cabeçalhos de coluna
16
17 // gera o valor de cada elemento do array na saída
18 for (int counter = 0; counter < array.length; counter++)
19 System.out.printf("%5d%8d\n", counter, array[counter]);
20 } // fim de main
21 } // fim da classe InitArray

```

| Índice | Valor |
|--------|-------|
| 0      | 2     |
| 1      | 4     |
| 2      | 6     |
| 3      | 8     |
| 4      | 10    |
| 5      | 12    |
| 6      | 14    |
| 7      | 16    |
| 8      | 18    |
| 9      | 20    |

**Figura E.4** Calculando os valores a serem colocados nos elementos de um array.

A linha 8 usa o modificador `final` para declarar a variável constante `ARRAY_LENGTH` com o valor 10. As variáveis constantes devem ser inicializadas antes de serem usadas e, depois disso, não podem ser modificadas. Se você tentar *modificar* uma variável `final` depois de ser inicializada em sua declaração, o compilador gerará uma mensagem de erro como

cannot assign a value to final variable *nomeDaVariável*

Se for feita uma tentativa de acessar o valor de uma variável `final` antes que ela seja inicializada, o compilador gerará uma mensagem de erro como

variable *nomeDaVariável* might not have been initialized



### Boa prática de programação E.1

As variáveis constantes também são chamadas de **constantes nomeadas**. Frequentemente, elas tornam os programas mais legíveis do que os que usam valores literais (por exemplo, 10) – uma constante nomeada como `ARRAY_LENGTH` indica claramente sua finalidade, enquanto um valor literal pode ter diferentes significados de acordo com seu contexto.



### Usando gráficos de barras para exibir dados de array

Muitos programas apresentam dados para os usuários em forma de gráficos. Por exemplo, os valores numéricos são frequentemente exibidos como barras em um gráfico de barras. Em um gráfico assim, as barras representam proporcionalmente os valores numéricos maiores. Uma maneira simples de exibir dados numéricos graficamente é com um gráfico de barras que mostre cada valor numérico como uma barra de asteriscos (\*).

Muitas vezes, os professores gostam de examinar a distribuição das notas de uma prova. Um professor poderia representar em um gráfico o número de notas em cada uma de várias categorias para visualizar a distribuição das notas. Suponha que as notas de um exame foram 87, 68, 94, 100, 83, 78, 85, 91, 76 e 87. Elas incluem uma nota 100, duas na casa dos 90, quatro na casa dos 80, duas na casa dos 70, uma na casa dos 60 e nenhuma abaixo de 60. Nosso próximo aplicativo (Fig. E.5) armazena esses dados de distribuição de notas em um array de 11 elementos, cada um correspondendo a uma categoria de notas. Por exemplo, array[0] indica o número de notas no intervalo de 0 a 9, array[7] o número de notas no intervalo de 70 a 79 e array[10] o número de notas 100.

```

1 // Fig. E.5: BarChart.java
2 // Programa para impressão de gráfico de barras.
3
4 public class BarChart
5 {
6 public static void main(String[] args)
7 {
8 int[] array = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };
9
10 System.out.println("Grade distribution:");
11
12 // para cada elemento do array, gera uma barra do gráfico
13 for (int counter = 0; counter < array.length; counter++)
14 {
15 // gera o rótulo da barra ("00-09: ", ..., "90-99: ", "100: ")
16 if (counter == 10)
17 System.out.printf("%5d: ", 100);
18 else
19 System.out.printf("%02d-%02d: ",
20 counter * 10, counter * 10 + 9);
21
22 // imprime a barra de asteriscos
23 for (int stars = 0; stars < array[counter]; stars++)
24 System.out.print("*");
25
26 System.out.println(); // começa uma nova linha de saída
27 } // fim do for externo
28 } // fim de main
29 } // fim da classe BarChart

```

```

Grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
100: *

```

**Figura E.5** Programa para impressão de gráfico de barras.

O aplicativo lê os números do array e representa a informação como um gráfico de barras. Ele exibe cada intervalo de notas seguido por uma barra de asteriscos indicando o número de notas nesse intervalo. Para rotular cada barra, as linhas 16 a 20 geram um intervalo de notas (por exemplo, "70-79: ") com base no valor atual de counter. Quando counter é 10, a linha 17 gera 100 com largura de campo 5, seguido de dois pontos e um espaço para alinhar o rótulo "100: " com os rótulos das outras barras. A instrução for aninhada (linhas 23 e 24) gera as barras. Observe a condição de continuação do loop na linha 23 (stars < array[counter]). Sempre que o programa chega ao for interno, o loop conta de 0 a array[counter], usando assim um valor do array para determinar o número de asteriscos a exibir. Nesse exemplo, nenhum aluno recebeu uma nota abaixo de 60, de modo que array[0] a array[5] contém zeros e nenhum asterisco é exibido ao lado dos seis primeiros intervalos de nota. Na linha 19, o especificador de formato %02d indica que um valor int deve ser formatado como um campo de dois dígitos. O **flag 0** no especificador de formato exibe um 0 à esquerda para valores com menos dígitos que o campo width (2).

### Usando os elementos de um array como contadores

Às vezes, os programas usam variáveis contadoras para resumir dados, como nos resultados de um levantamento. A Figura E.6 usa o array frequency (linha 10) para contar as ocorrências de cada lado de um dado lançado 6.000.000 vezes. A linha 14 usa o valor aleatório para determinar qual elemento de frequency vai incrementar durante cada iteração do loop. O cálculo na linha 14 produz números aleatórios de 1 a 6; portanto, o array frequency deve ser grande o bastante para armazenar seis contadores. Contudo, usamos um array de sete elementos no qual ignoramos frequency[0] – é mais lógico fazer o valor de face 1 incrementar frequency[1] do que frequency[0]. Assim, cada valor de face é usado como índice para o array frequency. Na linha 14, o cálculo dentro dos colchetes é avaliado primeiro para determinar qual elemento do array vai ser incrementado e, então, o operador ++ soma um a esse elemento. As linhas 19 e 20 fazem um loop pelo array frequency para gerar os resultados na saída.

```

1 // Fig. E.7: RollDie.java
2 // Programa de lançamento de dado usando arrays em vez de switch.
3 import java.util.Random;
4
5 public class RollDie
6 {
7 public static void main(String[] args)
8 {
9 Random randomNumbers = new Random(); // gerador de números aleatórios
10 int[] frequency = new int[7]; // array de contadores de frequência
11
12 // lança o dado 6.000.000 vezes; usa o valor do dado como índice de frequency
13 for (int roll = 1; roll <= 6000000; roll++)
14 ++frequency[1 + randomNumbers.nextInt(6)];
15
16 System.out.printf("%s%10s\n", "Face", "Frequency");
17
18 // gera o valor de cada elemento do array na saída
19 for (int face = 1; face < frequency.length; face++)
20 System.out.printf("%4d%10d\n", face, frequency[face]);
21 } // fim de main
22 } // fim da classe RollDie

```

**Figura E.6** Programa de lançamento de dado usando arrays em vez de switch. (continua)

| Face | Frequency |
|------|-----------|
| 1    | 999690    |
| 2    | 999512    |
| 3    | 1000575   |
| 4    | 999815    |
| 5    | 999781    |
| 6    | 1000627   |

**Figura E.6** Programa de lançamento de dado usando arrays em vez de switch.

### Usando arrays para analisar resultados de levantamento

Nosso próximo exemplo usa arrays para resumir dados coletados em um levantamento. Considere o enunciado de problema a seguir:

*Pediram a 20 alunos para que classificassem, em uma escala de 1 a 5, a qualidade dos alimentos na lanchonete da escola, sendo 1 “péssima” e 5 “excelente”. Coloque as 20 respostas em um array inteiro e determine a frequência de cada classificação.*

Esse é um aplicativo de processamento de array típico (Fig. E.7). Queremos resumir o número de respostas de cada tipo (isto é, de 1 a 5). O array `responses` (linhas 9 e 10) é um array inteiro de 20 elementos contendo as respostas dadas pelos alunos no levantamento. O último valor no array é intencionalmente uma resposta incorreta (14). Quando um programa Java é executado, é verificada a validade dos índices dos elementos do array – todos os índices devem ser maiores ou iguais a 0 e menores que o comprimento do array. Qualquer tentativa de acessar um elemento fora desse intervalo de índices resulta em um erro de tempo de execução conhecido como `ArrayIndexOutOfBoundsException`. No final desta seção, vamos discutir o valor de respostas inválidas, demonstrar a **verificação de limites** dos arrays e apresentar o mecanismo de tratamento de exceções do Java, o qual pode ser usado para detectar e tratar uma exceção `ArrayIndexOutOfBoundsException`.

```

1 // Fig. E.7: StudentPoll.java
2 // Programa de análise de pesquisa de opinião.
3
4 public class StudentPoll
5 {
6 public static void main(String[] args)
7 {
8 // array de respostas dos alunos (normalmente inseridas em tempo de execução)
9 int[] responses = { 1, 2, 5, 4, 3, 5, 2, 1, 3, 3, 1, 4, 3, 3, 3,
10 2, 3, 3, 2, 14 };
11 int[] frequency = new int[6]; // array de contadores de frequência
12
13 // para cada resposta, seleciona o elemento responses e utiliza esse valor
14 // como índice de frequency para determinar o elemento a ser incrementado
15 for (int answer = 0; answer < responses.length; answer++)
16 {
17 try
18 {
19 ++frequency[responses[answer]];
20 } // fim do try
21 catch (ArrayIndexOutOfBoundsException e)
22 {
23 System.out.println(e);
24 }
25 }
26 }
27 }

```

**Figura E.7** Programa de análise de pesquisa de opinião. (continua)

```

24 System.out.printf(" responses[%d] = %d\n\n",
25 answer, responses[answer]);
26 } // fim do catch
27 } // fim do for
28
29 System.out.printf("%s%10s\n", "Rating", "Frequency");
30
31 // gera o valor de cada elemento do array na saída
32 for (int rating = 1; rating < frequency.length; rating++)
33 System.out.printf("%6d%10d\n", rating, frequency[rating]);
34 } // fim de main
35 } // fim da classe StudentPoll

```

```

java.lang.ArrayIndexOutOfBoundsException: 14
responses[19] = 14
Rating Frequency
1 3
2 4
3 8
4 2
5 2

```

**Figura E.7** Programa de análise de pesquisa de opinião.

### O array frequency

Usamos o array `frequency` de *seis elementos* (linha 11) para contar o número de ocorrências de cada resposta. Cada elemento é usado como contador para um dos tipos possíveis de respostas – `frequency[1]` conta o número de alunos que classificaram os alimentos como 1, `frequency[2]` conta o número de alunos que classificaram os alimentos como 2 e assim por diante.

### Resumindo os resultados

A instrução `for` (linhas 15 a 27) lê as respostas do array `responses` uma por vez e incrementa um dos contadores, de `frequency[1]` a `frequency[5]` – ignoramos `frequency[0]` porque as respostas são limitadas ao intervalo de 1 a 5. A principal instrução do loop aparece na linha 19. Essa instrução incrementa o contador `frequency` apropriado, conforme determinado pelo valor de `responses[answer]`. Vamos percorrer as primeiras iterações da instrução `for`:

- Quando o contador `answer` é 0, `responses[answer]` é o valor de `responses[0]` (isto é, 1 – consulte a linha 9). Nesse caso, `frequency[responses[answer]]` é interpretado como `frequency[1]` e o contador `frequency[1]` é incrementado por um. Para avaliar a expressão, começamos com o valor no conjunto *mais interno* de colchetes (`answer`, atualmente 0). O valor de `answer` é acrescentado à expressão e o próximo conjunto de colchetes (`responses[answer]`) é avaliado. Esse valor é usado como índice do array `frequency` para determinar qual contador deve ser incrementado (neste caso, `frequency[1]`).
- Na próxima passagem pelo loop, `answer` é 1, `responses[answer]` é o valor de `responses[1]` (isto é, 2 – consulte a linha 9); portanto, `frequency[responses[answer]]` é interpretado como `frequency[2]`, fazendo `frequency[2]` ser incrementado.
- Quando `answer` é 2, `responses[answer]` é o valor de `responses[2]` (isto é, 5 – consulte a linha 9); portanto, `frequency[responses[answer]]` é interpretado como `frequency[5]`, fazendo `frequency[5]` ser incrementado e assim por diante.

Independentemente do número de respostas processadas no levantamento, apenas um array de seis elementos (no qual ignoramos o elemento zero) é exigido para resumir os resultados, pois todos os valores de resposta corretos estão entre 1 e 5, e os valores de índice para um array de seis elementos são de 0 a 5. Na saída do programa, a coluna Frequency resume somente 19 dos 20 valores do array responses – o último elemento do array contém uma resposta incorreta que não foi contada.

### **Tratamento de exceção: processando a resposta incorreta**

Uma **exceção** indica um problema que ocorre enquanto um programa é executado. O nome “exceção” sugere que o problema ocorre raramente – se a “regra” é que uma instrução seja executada corretamente, então o problema representa a “exceção à regra”. O **tratamento de exceções** permite que você crie **programas tolerantes à falha** que podem resolver (ou tratar) exceções. Em muitos casos, isso permite que um programa continue a ser executado como se não fosse encontrado um problema. Por exemplo, o aplicativo StudentPoll ainda exibe resultados (Fig. E.7), mesmo uma das respostas estando fora do intervalo. Problemas mais sérios podem impedir que um programa continue a ser executado normalmente, em vez de exigir que ele notifique o usuário sobre o problema e, então, termine. Quando a JVM ou um método detecta um problema, como um índice de array inválido ou um argumento de método inválido, **lança** uma exceção – isto é, ocorre uma exceção.

### **A instrução try**

Para tratar uma exceção, coloque qualquer código que possa lançar uma exceção em uma **instrução try** (linhas 17 a 26). O **bloco try** (linhas 17 a 20) contém o código que pode **lançar** uma exceção, e o **bloco catch** (linhas 21 a 26) contém o código que *trata* a exceção, caso ocorra. É possível haver muitos blocos catch para tratar diferentes tipos de exceções que possam ser lançadas no bloco try correspondente. Quando a linha 19 incrementa corretamente um elemento do array frequency, as linhas 21 a 26 são ignoradas. As chaves que delimitam os corpos dos blocos try e catch são obrigatórias.

### **Executando o bloco catch**

Quando o programa encontra o valor 14 no array responses, ele tenta somar 1 a frequency[14], que está *fora* dos limites do array – o array frequency tem apenas seis elementos. Como a verificação de limites do array é feita em tempo de execução, a JVM gera uma exceção – especificamente, a linha 19 lança uma exceção **ArrayIndexOutOfBoundsException** para notificar o programa sobre esse problema. Nesse ponto, o bloco try termina e o bloco catch começa a ser executado – se você declarou quaisquer variáveis no bloco try, agora elas estão fora do escopo e são inacessíveis no bloco catch.

O bloco catch declara um tipo (IndexOutOfRangeException) e um parâmetro de exceção (e). Esse bloco pode tratar exceções do tipo especificado. Dentro do bloco catch, você pode usar o identificador do parâmetro para interagir com um objeto de exceção capturado.



#### **Dica de prevenção de erro E.1**

*Ao escrever um código para acessar um elemento de array, certifique-se de que o índice do array permaneça maior ou igual a 0 e menor que o comprimento do array. Isso ajuda a evitar a exceção ArrayIndexOutOfBoundsException em seu programa.*

### Método `toString` do parâmetro de exceção

Quando as linhas 21 a 26 *capturam* a exceção, o programa exibe uma mensagem indicando o problema ocorrido. A linha 23 chama implicitamente o método `toString` do objeto de exceção para obter a mensagem de erro armazenada no objeto de exceção e exibi-la. Uma vez exibida a mensagem neste exemplo, a exceção é considerada tratada e o programa continua com a próxima instrução após a chave de fechamento do bloco `catch`. Neste exemplo, o final da instrução `for` é atingido (linha 27), de modo que o programa continua com o incremento da variável de controle na linha 15. Usamos tratamento de exceção novamente no Apêndice F, e o Apêndice H apresenta um exame mais aprofundado sobre esse assunto.

## E.5 Estudo de caso: simulação de embaralhamento e distribuição de cartas

Até aqui, os exemplos desde apêndice usaram arrays contendo elementos de tipos primitivos. Lembre-se da seção E.2, em que os elementos de um array podem ser tipos primitivos ou tipos de referência. Esta seção utiliza geração de números aleatórios e um array de elementos tipo de referência – a saber, objetos representando cartas de baralho – para desenvolver uma classe que simula o embaralhamento e a distribuição de cartas. Essa classe pode então ser usada a fim de implementar aplicativos para jogos de baralho específicos.

Primeiro, desenvolvemos a classe `Card` (Fig. E.8), a qual representa uma carta com uma face (por exemplo, "Ás", "Dois", "Três", ..., "Valete", "Dama", "Rei") e um naipe (por exemplo, "Copas", "Ouros", "Paus", "Espadas"). Em seguida, desenvolvemos a classe `DeckOfCards` (Fig. E.9), a qual cria um baralho de 52 cartas no qual cada elemento é um objeto `Card`. Então, construímos um aplicativo de teste (Fig. E.10) que demonstra os recursos de embaralhamento e distribuição de cartas da classe `DeckOfCards`.

### Classe `Card`

A classe `Card` (Fig. E.8) contém duas variáveis de instância `String` – `face` e `suit` –, utilizadas para armazenar referências para o nome da face e para o nome do naipe de um objeto `Card` específico. O construtor da classe (linhas 10 a 14) recebe duas `Strings`, as quais utiliza para inicializar `face` e `suit`. O método `toString` (linhas 17 a 20) cria uma `String` consistindo na face da carta, a `String` " of " e o naipe da carta. O método `toString` de `Card` pode ser chamado explicitamente para obter uma representação de `string` de um objeto `Card` (por exemplo, "Ace of Spades"). O método `toString` de um objeto é chamado *implicitamente* quando o objeto é usado onde uma `String` é esperada (por exemplo, quando `printf` gera na saída o objeto como uma `String` com o especificador de formato `%s` ou quando o objeto é concatenado a uma `String` com o operador `+`). Para que esse comportamento ocorra, `toString` deve ser declarado com o cabeçalho mostrado na Fig. E.8.

```

1 // Fig. E.8: Card.java
2 // A classe Card representa uma carta de baralho.
3
4 public class Card
5 {
6 private String face; // face da carta ("Ás", "Dois", ...)
7 private String suit; // naipe da carta ("Copas", "Ouros", ...)
8
9 // o construtor de dois argumentos inicializa a face e o naipe da carta

```

**Figura E.8** Classe `Card` que representa uma carta de baralho. (continua)

```

10 public Card(String cardFace, String cardSuit)
11 {
12 face = cardFace; // inicializa a face da carta
13 suit = cardSuit; // inicializa o naipe da carta
14 } // fim do construtor Card de dois argumentos
15
16 // retorna representação de String de Card
17 public String toString()
18 {
19 return face + " of " + suit;
20 } // fim do método toString
21 } // fim da classe Card

```

**Figura E.8** Classe Card que representa uma carta de baralho.

### Classe DeckOfCards

A classe DeckOfCards (Fig. E.9) declara como variável de instância um array Card chamado deck (linha 7). Um array de tipo de referência é declarado como qualquer outro array. A classe DeckOfCards também declara uma variável de instância inteira currentCard (linha 8), representando o próximo objeto Card a ser distribuído a partir do array deck, e uma constante nomeada NUMBER\_OF\_CARDS (linha 9) indicando o número de objetos Card no baralho (52).

```

1 // Fig. E.9: DeckOfCards.java
2 // A classe DeckOfCards representa as cartas de um baralho.
3 import java.util.Random;
4
5 public class DeckOfCards
6 {
7 private Card[] deck; // array de objetos Card
8 private int currentCard; // índice do próximo objeto Card a ser distribuído (0-51)
9 private static final int NUMBER_OF_CARDS = 52; // constante do nº de objetos Card
10 // gerador de números aleatórios
11 private static final Random randomNumbers = new Random();
12
13 // o construtor preenche o baralho de cartas
14 public DeckOfCards()
15 {
16 String[] faces = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
17 "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
18 String[] suits = { "Hearts", "Diamonds", "Clubs", "Spades" };
19
20 deck = new Card[NUMBER_OF_CARDS]; // cria array de objetos Card
21 currentCard = 0; // configura currentCard de modo que o primeiro
22 // objeto Card distribuído seja deck[0]
23
24 // preenche o baralho com objetos Card
25 for (int count = 0; count < deck.length; count++)
26 deck[count] =
27 new Card(faces[count % 13], suits[count / 13]);
28 } // fim do construtor DeckOfCards
29
30 // embaralha as cartas com algoritmo de uma passagem
31 public void shuffle()
32 {
33 // após embaralhar, a distribuição deve começar novamente em deck[0]
34 currentCard = 0; // reinicializa currentCard
35
36 // para cada Card, escolhe outro Card aleatório (de 0 a 51) e os troca

```

**Figura E.9** A classe DeckOfCards representa as cartas de um baralho.(continua)

```

36 for (int first = 0; first < deck.length; first++)
37 {
38 // seleciona um número aleatório entre 0 e 51
39 int second = randomNumbers.nextInt(NUMBER_OF_CARDS);
40
41 // troca Card atual por Card selecionado aleatoriamente
42 Card temp = deck[first];
43 deck[first] = deck[second];
44 deck[second] = temp;
45 } // fim do for
46 } // fim do método shuffle
47
48 // distribui uma carta
49 public Card dealCard()
50 {
51 // determina se cartas continuam a ser distribuídas
52 if (currentCard < deck.length)
53 return deck[currentCard++]; // retorna Card atual no array
54 else
55 return null; // retorna null para indicar que todas as cartas foram distribuídas
56 } // fim do método dealCard
57 } // fim da classe DeckOfCards

```

**Figura E.9** A classe `DeckOfCards` representa as cartas de um baralho.

### Construtor de `DeckOfCards`

O construtor da classe instancia o array `deck` (linha 20) com `NUMBER_OF_CARDS` (52), elementos que são todos `null` por padrão. As linhas 24 a 26 preenchem o baralho `deck` com objetos `Card`. O loop inicializa a variável de controle `count` com 0 e se repete enquanto `count` for menor que `deck.length`, fazendo que `count` assumira cada valor inteiro de 0 a 51 (os índices do array `deck`). Cada `Card` é instanciado e inicializado com uma `String` do array `faces` (o qual contém de "Ace" até "King") e uma `String` do array `suits` (o qual contém "Hearts", "Diamonds", "Clubs" e "Spades"). O cálculo `count % 13` resulta sempre em um valor de 0 a 12 (os 13 índices do array `faces` nas linhas 16 e 17), e o cálculo `count / 13` resulta sempre em um valor de 0 a 3 (os quatro índices do array `suits` na linha 18). Quando o array `deck` é inicializado, ele contém cartas com faces de "Ace" a "King", em ordem para cada naipe ("Hearts", depois "Diamonds", depois "Clubs", depois "Spades").

### Método `shuffle` de `DeckOfCards`

O método `shuffle` (linhas 30 a 46) embaralha as cartas do baralho. Ele faz um loop por todas as 52 cartas. Para cada carta é escolhido um número aleatório entre 0 e 51 a fim de selecionar outra carta; então, a carta atual e a selecionada aleatoriamente são trocadas no array. Essa troca é feita pelas atribuições nas linhas 42 a 44. A variável extra `temp` armazena temporariamente um dos dois objetos `Card` que estão sendo trocados. A troca não pode ser feita apenas com as duas instruções

```

deck[first] = deck[second];
deck[second] = deck[first];

```

Se `deck[first]` é "Ace" de "Spades" e `deck[second]` é "Queen" de "Hearts", após a primeira atribuição, os dois elementos do array contém "Queen" de "Hearts", e "Ace" de "Spades" é perdido — por isso, a variável extra `temp` é necessária. Depois que o loop for termina, os objetos `Card` são ordenados aleatoriamente. No total, apenas 52 trocas são feitas em uma única passagem pelo array inteiro, e o array de objetos `Card` está embaralhado! [Obs.: recomenda-se usar o chamado algoritmo de embaralhamento imparcial para jogos de baralho reais. Tal algoritmo garante que todas as sequências possíveis de



cartas embaralhadas tenham a mesma probabilidade de ocorrer. Um algoritmo de embaralhamento imparcial popular é o algoritmo de Fisher-Yates.]

### **Método `dealCard` de `DeckOfCards`**

O método `dealCard` (linhas 49 a 56) distribui um objeto `Card` do array. Lembre-se de que `currentCard` indica o índice do próximo objeto `Card` a ser distribuído (isto é, o objeto `Card` no topo do baralho). Assim, a linha 52 compara `currentCard` com o comprimento do array. Se `deck` não está vazio (isto é, `currentCard` é menor que 52), a linha 53 retorna o objeto `Card` do “topo” e pós-incrementa `currentCard` a fim de se preparar para a próxima chamada de `dealCard` – caso contrário, `null` é retornado.

### **Embaralhando e distribuindo cartas**

A Figura E.10 demonstra a classe `DeckOfCards` (Fig. E.9). A linha 9 cria um objeto `DeckOfCards` chamado `myDeckOfCards`. O construtor de `DeckOfCards` cria o baralho com 52 objetos `Card` em ordem, por naipe e face. A linha 10 chama o método `shuffle` de `myDeckOfCards` para reordenar os objetos `Card`. As linhas 13 a 20 distribuem todos os 52 objetos `Card` e os imprimem em quatro colunas de 13 cartas cada uma. A linha 16 distribui um objeto `Card` chamando o método `dealCard` de `myDeckOfCards` e, então, o exibe justificando à esquerda em um campo de 19 caracteres. Quando um objeto `Card` é gerado como uma `String`, o método `toString` de `Card` (linhas 17 a 20 da Fig. E.8) é chamado implicitamente. As linhas 18 e 19 (Fig. E.10) iniciam uma nova linha após cada quatro objetos `Card`.

```

1 // Fig. E.10: DeckOfCardsTest.java
2 // Embaralhamento e distribuição de cartas.
3
4 public class DeckOfCardsTest
5 {
6 // executa o aplicativo
7 public static void main(String[] args)
8 {
9 DeckOfCards myDeckOfCards = new DeckOfCards();
10 myDeckOfCards.shuffle(); // coloca as cartas em ordem aleatória
11
12 // imprime todas as 52 cartas na ordem em que são distribuídas
13 for (int i = 1; i <= 52; i++)
14 {
15 // distribui e exibe uma carta
16 System.out.printf("%-19s", myDeckOfCards.dealCard());
17
18 if (i % 4 == 0) // gera uma nova linha na saída depois de cada quarta carta
19 System.out.println();
20 } // fim do for
21 } // fim de main
22 } // fim da classe DeckOfCardsTest

```

|                   |                   |                   |                  |
|-------------------|-------------------|-------------------|------------------|
| Six of Spades     | Eight of Spades   | Six of Clubs      | Nine of Hearts   |
| Queen of Hearts   | Seven of Clubs    | Nine of Spades    | King of Hearts   |
| Three of Diamonds | Deuce of Clubs    | Ace of Hearts     | Ten of Spades    |
| Four of Spades    | Ace of Clubs      | Seven of Diamonds | Four of Hearts   |
| Three of Clubs    | Deuce of Hearts   | Five of Spades    | Jack of Diamonds |
| King of Clubs     | Ten of Hearts     | Three of Hearts   | Six of Diamonds  |
| Queen of Clubs    | Eight of Diamonds | Deuce of Diamonds | Ten of Diamonds  |
| Three of Spades   | King of Diamonds  | Nine of Clubs     | Six of Hearts    |
| Ace of Spades     | Four of Diamonds  | Seven of Hearts   | Eight of Clubs   |
| Deuce of Spades   | Eight of Hearts   | Five of Hearts    | Queen of Spades  |
| Jack of Hearts    | Seven of Spades   | Four of Clubs     | Nine of Diamonds |
| Ace of Diamonds   | Queen of Diamonds | Five of Clubs     | King of Spades   |
| Five of Diamonds  | Ten of Clubs      | Jack of Spades    | Jack of Clubs    |

**Figura E.10** Embaralhamento e distribuição de cartas.

## E.6 Instrução for melhorada

A **instrução for melhorada** itera pelos elementos de um array *sem* usar um contador, evitando assim a possibilidade de “dar um passo fora” do array. Mostramos como usar a instrução for melhorada com as estruturas de dados predefinidas (denominadas coleções) da API Java na seção E.12. A sintaxe de uma instrução for melhorada é:

```
for (parâmetro : nomeDoArray)
 instrução
```

onde *parâmetro* tem um tipo e um identificador (por exemplo, `int number`) e *nomeDoArray* é o array pelo qual se vai iterar. O tipo do parâmetro deve ser coerente com o tipo dos elementos do array. Como ilustra o próximo exemplo, o identificador representa valores de elemento sucessivos no array, em sucessivas iterações do loop.

A Figura E.11 usa a instrução for melhorada (linhas 12 e 13) para somar os valores inteiros em um array de notas de alunos. O parâmetro do for melhorado é de tipo `int`, pois o array contém valores `int` – o loop seleciona um valor `int` do array durante cada iteração. A instrução for melhorada itera pelos sucessivos valores do array, um por um. O cabeçalho da instrução pode ser lido como “para cada iteração, atribui o próximo elemento do array à variável `int number` e, então, executa a instrução seguinte”. Assim, para cada iteração, o identificador `number` representa um valor `int` no array. As linhas 12 e 13 são equivalentes à seguinte instrução de repetição controlada por contador, exceto que `counter` não pode ser acessada no corpo da instrução for melhorada:

```
for (int counter = 0; counter < array.length; counter++)
 total += array[counter];
```

```
1 // Fig. E.11: EnhancedForTest.java
2 // Usando a instrução for melhorada para totalizar valores inteiros em um array.
3
4 public class EnhancedForTest
5 {
6 public static void main(String[] args)
7 {
8 int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9 int total = 0;
10
11 // soma o valor de cada elemento a total
12 for (int number : array)
13 total += number;
14
15 System.out.printf("Total of array elements: %d\n", total);
16 } // fim de main
17 } // fim da classe EnhancedForTest
```

```
Total of array elements: 849
```

**Figura E.11** Usando a instrução for melhorada para totalizar valores inteiros em um array.

A instrução for melhorada simplifica o código para iterar por um array. Observe, contudo, que *a instrução for melhorada só pode ser usada para obter elementos de array – ela não pode ser usada para modificar elementos*. Se seu programa precisa modificar elementos, use a instrução for controlada por contador tradicional.

A instrução `for` melhorada pode ser usada no lugar da instrução `for` controlada por contador quando o código que faz loop por um array *não* exige acesso ao contador que indica o índice do elemento atual do array. Por exemplo, totalizar os valores inteiros de um array exige acesso somente aos valores dos elementos – o índice de cada elemento é irrelevante. Contudo, se um programa precisar de um contador por algum motivo que não seja simplesmente fazer loop por um array (por exemplo, para imprimir um número de índice ao lado de cada valor de elemento do array, como nos exemplos anteriores deste apêndice), use a instrução `for` controlada por contador.

## E.7 Passagem de arrays para métodos

Esta seção demonstra como passar arrays e elementos individuais de array como argumentos para métodos. Para passar um argumento de array para um método, especifique o nome do array sem colchetes. Por exemplo, se o array `hourlyTemperatures` é declarado como

```
double[] hourlyTemperatures = new double[24];
```

então a chamada de método

```
modifyArray(hourlyTemperatures);
```

passa a referência do array `hourlyTemperatures` para o método `modifyArray`. Todo objeto array “conhece” seu próprio comprimento (por meio de seu campo `length`). Assim, quando passamos a referência de um objeto array para um método, não precisamos passar o comprimento do array como um argumento adicional.

Para que um método receba a referência de um array por meio de uma chamada, a lista de parâmetros do método deve especificar um parâmetro de array. Por exemplo, o cabeçalho do método `modifyArray` poderia ser escrito como

```
void modifyArray(double[] b)
```

indicando que `modifyArray` recebe a referência de um array `double` no parâmetro `b`. A chamada de método passa a referência do array `hourlyTemperature`, de modo que, quando o método chamado utiliza a variável de array `b`, ela *se refere* ao mesmo objeto array que `hourlyTemperatures` no chamador.

Quando um argumento de um método é um array inteiro ou um elemento individual de um array de um tipo de referência, o método chamado recebe uma *cópia* da referência. Contudo, quando um argumento de um método é um elemento individual de um array de um tipo primitivo, o método chamado recebe uma cópia do *valor* do elemento. Esses valores primitivos são chamados de **escalares** ou **quantidades escalares**. Para passar um elemento individual de um array para um método, use o nome indexado do elemento do array como argumento na chamada de método.

A Figura E.12 demonstra a diferença entre passar um array inteiro e passar um elemento de array de tipo primitivo para um método. Observe que `main` chama os métodos estáticos `modifyArray` (linha 19) e `modifyElement` (linha 30) diretamente. Lembre-se da seção D.4, em que um método estático de uma classe pode chamar outros métodos estáticos da mesma classe diretamente.

```

1 // Fig. E.12: PassArray.java
2 // Passando arrays e elementos individuais de array para métodos.
3
4 public class PassArray
5 {
6 // main cria o array e chama modifyArray e modifyElement
7 public static void main(String[] args)
8 {
9 int[] array = { 1, 2, 3, 4, 5 };
10
11 System.out.println(
12 "Effects of passing reference to entire array:\n" +
13 "The values of the original array are:");
14
15 // gera os elementos originais do array
16 for (int value : array)
17 System.out.printf(" %d", value);
18
19 modifyArray(array); // passa referência de array
20 System.out.println("\n\nThe values of the modified array are:");
21
22 // gera os elementos do array modificados
23 for (int value : array)
24 System.out.printf(" %d", value);
25
26 System.out.printf(
27 "\n\nEffects of passing array element value:\n" +
28 "array[3] before modifyElement: %d\n", array[3]);
29
30 modifyElement(array[3]); // tenta modificar array[3]
31 System.out.printf(
32 "array[3] after modifyElement: %d\n", array[3]);
33 } // fim de main
34
35 // multiplica cada elemento de um array por 2
36 public static void modifyArray(int[] array2)
37 {
38 for (int counter = 0; counter < array2.length; counter++)
39 array2[counter] *= 2;
40 } // fim do método modifyArray
41
42 // multiplica argumento por 2
43 public static void modifyElement(int element)
44 {
45 element *= 2;
46 System.out.printf(
47 "Value of element in modifyElement: %d\n", element);
48 } // fim do método modifyElement
49 } // fim da classe PassArray

```

Effects of passing reference to entire array:  
The values of the original array are:  
1 2 3 4 5

The values of the modified array are:  
2 4 6 8 10

Effects of passing array element value:  
array[3] before modifyElement: 8  
Value of element in modifyElement: 16  
array[3] after modifyElement: 8

**Figura E.12** Passando arrays e elementos individuais de array para métodos.

A instrução `for` melhorada nas linhas 16 e 17 gera os cinco elementos `int` do array. A linha 19 chama o método `modifyArray`, passando `array` como argumento. O método `modifyArray` (linhas 36 a 40) recebe uma cópia da referência de array e usa a referência para multiplicar cada um dos elementos de array por 2. Para provar que os elementos de array foram modificados, as linhas 23 e 24 geram os cinco elementos de array na saída novamente. Como a saída mostra, o método `modifyArray` duplicou o valor de cada elemento. Poderíamos não usar a instrução `for` melhorada nas linhas 38 e 39, pois estávamos modificando os elementos do array.

A Figura E.12 demonstra em seguida que, quando uma cópia de um elemento individual de um array de tipo primitivo é passada para um método, modificar a *cópia* no método chamado *não* afeta o valor original desse elemento no array do método chamador. As linhas 26 a 28 geram o valor de `array[3]` antes de chamar o método `modifyElement`. Lembre-se de que o valor desse elemento agora é 8, após ser modificado na chamada de `modifyArray`. A linha 30 chama o método `modifyElement` e passa `array[3]` como argumento. Lembre-se de que `array[3]` é na verdade um único valor `int` (8) no array. Portanto, o programa passa uma cópia do valor de `array[3]`. O método `modifyElement` (linhas 43 a 48) multiplica o valor recebido como argumento por 2, armazena o resultado em seu parâmetro `element` e, então, gera o valor de `element` (16). Como os parâmetros de método, da mesma forma que as variáveis locais, deixam de existir quando o método no qual são declarados conclui a execução, o parâmetro de método `element` é destruído quando o método `modifyElement` termina. Quando o programa retorna o controle para `main`, as linhas 31 e 32 geram o valor *intacto* de `array[3]` (isto é, 8).

### Observações sobre a passagem de argumentos para métodos

O exemplo anterior demonstrou como os arrays e elementos de array de tipo primitivo são passados como argumentos para métodos. Agora, examinamos mais detidamente como os argumentos em geral são passados para métodos. Em muitas linguagens de programação, duas maneiras de passar argumentos em chamadas de método são a **passagem por valor** e a **passagem por referência** (também chamadas de **chamada por valor** e **chamada por referência**). Quando um argumento é passado por valor, uma cópia do *valor* do argumento é passada para o método chamado. O método chamado trabalha exclusivamente com a cópia. Alterações na cópia do método chamado *não* afetam o valor original da variável no chamador.

Quando um argumento é passado por referência, o método chamado pode acessar diretamente o valor do argumento no chamador e modificar esse dado, se necessário. A passagem por referência aumenta o desempenho por eliminar a necessidade de copiar volumes de dados possivelmente grandes.

Ao contrário de algumas outras linguagens, o Java *não* permite que você escolha passagem por valor ou passagem por referência – *todos os argumentos são passados por valor*. Uma chamada de método pode passar dois tipos de valores para um método: cópias de valores primitivos (por exemplo, valores de tipo `int` e `double`) e cópias de referências para objetos. Objetos em si não podem ser passados para métodos. Quando um método modifica um parâmetro de tipo primitivo, alterações feitas no parâmetro não têm efeito sobre o valor original do argumento no método chamador. Por exemplo, quando a linha 30 em `main` da Fig. E.12 passa `array[3]` para o método `modifyElement`, a instrução na linha 45, que duplica o valor do parâmetro `element`, *não* tem efeito sobre o valor de `array[3]` em `main`. Isso também vale para parâmetros de tipo de referência. Se você modifica um parâmetro de tipo de referência para que ele faça referência a outro objeto,

somente o parâmetro se refere ao novo objeto – a referência armazenada na variável do chamador ainda se refere ao objeto original.

Embora a referência de um objeto seja passada por valor, um método ainda pode interagir com o objeto referenciado, chamando seus métodos `public` usando a cópia da referência do objeto. Como a referência armazenada no parâmetro é uma cópia da referência que foi passada como argumento, o parâmetro no método chamado e o argumento no método chamador se referem ao mesmo objeto na memória. Por exemplo, na Fig. E.12, tanto o parâmetro `array2` no método `modifyArray` como a variável `array` em `main` se referem ao *mesmo* objeto `array` na memória. Quaisquer alterações feitas usando o parâmetro `array2` são realizadas no objeto a que `array` faz referência no método chamador. Na Fig. E.12, as alterações feitas em `modifyArray` usando `array2` afetam o conteúdo do objeto `array` referenciado por `array` em `main`. Assim, com uma referência para um objeto, o método chamado *pode* manipular o objeto do chamador diretamente.



#### Dica de desempenho E.1

*Passar arrays por referência faz sentido por questões de desempenho. Se os arrays fossem passados por valor, seria passada uma cópia de cada elemento. Para arrays grandes passados frequentemente, isso desperdiçaria tempo e consumiria armazenamento considerável para as cópias dos arrays.*

## E.8 Estudo de caso: classe `GradeBook` usando um array para armazenar as notas

As versões anteriores da classe `GradeBook` processam um conjunto de notas inseridas pelo usuário, mas não mantêm os valores de nota individuais em variáveis de instância da classe. Assim, cálculos repetidos exigem que o usuário redigite as mesmas notas. Uma maneira de resolver esse problema seria armazenar cada nota inserida em uma instância individual da classe. Por exemplo, poderíamos criar as variáveis de instância `grade1`, `grade2`, ..., `grade10` na classe `GradeBook` para armazenar 10 notas de aluno. Mas isso tornaria complicado o código para totalizar as notas e determinar a média da classe, e a classe não poderia processar mais do que 10 notas por vez. Resolvemos esse problema armazenando as notas em um array.

### **Armazenando notas de aluno em um array na classe `GradeBook`**

A classe `GradeBook` (Fig. E.13) usa um array de valores `int` para armazenar as notas de vários alunos obtidas em um único exame. Isso elimina a necessidade de inserir o mesmo conjunto de notas repetidamente. O array `grades` é declarado como uma variável de instância (linha 7), de modo que cada objeto `GradeBook` mantém seu próprio conjunto de notas. O construtor (linhas 10 a 14) tem dois parâmetros: o nome do curso e um array de notas. Quando um aplicativo (por exemplo, a classe `GradeBookTest` da Fig. E.14) cria um objeto `GradeBook`, passa um array `int` já existente para o construtor, o qual atribui a referência do array para a variável de instância `grades` (linha 13). O tamanho do array `grades` é determinado pelo comprimento do array passado para o construtor. Assim, um objeto `GradeBook` pode processar um número variável de notas. Os valores de nota no array passado poderiam ser inseridos por um usuário ou lidos de um arquivo no disco. Em nosso aplicativo de teste, inicializamos um array com valores de nota (Fig. E.14, linha 10). Uma vez armazenadas as notas na variável de instância `grades` da classe `GradeBook`,

todos os métodos da classe podem acessar os elementos de grades *com a frequência que for necessária* para efetuar vários cálculos.

O método `processGrades` (linhas 37 a 51) contém uma série de chamadas de método que geram um relatório resumindo as notas. A linha 40 chama o método `outputGrades` para imprimir o conteúdo do array `grades`. As linhas 134 a 136 no método `outputGrades` utilizam uma instrução `for` para gerar as notas dos alunos. Neste caso, um `for` controlado por contador *deve* ser usado, pois as linhas 135 e 136 utilizam o valor da variável contadora `student` para gerar na saída cada nota ao lado de um número de aluno em particular (consulte a saída na Fig. E.14). Embora os índices de array comecem em 0, normalmente um professor numeraria os alunos a partir de 1. Assim, as linhas 135 e 136 geram `student + 1` como número de aluno para produzir os rótulos de nota "Student 1: ", "Student 2: " e assim por diante.

```

1 // Fig. E.13: GradeBook.java
2 // Classe GradeBook usando um array para armazenar notas de teste.
3
4 public class GradeBook
5 {
6 private String courseName; // nome de curso dessa folha de notas
7 private int[] grades; // array de notas de aluno
8
9 // o construtor de dois argumentos inicializa courseName e o array grades
10 public GradeBook(String name, int[] gradesArray)
11 {
12 courseName = name; // inicializa courseName
13 grades = gradesArray; // armazena grades
14 } // fim do construtor de dois argumentos de GradeBook
15
16 // método para configurar o nome do curso
17 public void setCourseName(String name)
18 {
19 courseName = name; // armazena o nome do curso
20 } // fim do método setCourseName
21
22 // método para recuperar o nome do curso
23 public String getCourseName()
24 {
25 return courseName;
26 } // fim do método getCourseName
27
28 // exibe uma mensagem de boas-vindas para o usuário do aplicativo de notas
29 public void displayMessage()
30 {
31 // getCourseName recebe o nome do curso
32 System.out.printf("Welcome to the grade book for\n%s!\n\n",
33 getCourseName());
34 } // fim do método displayMessage
35
36 // efetua várias operações nos dados
37 public void processGrades()
38 {
39 // gera o array grades
40 outputGrades();
41
42 // chama o método getAverage para calcular a nota média
43 System.out.printf("\nClass average is %.2f\n", getAverage());

```

**Figura E.13** Classe `GradeBook` usando um array para armazenar notas de teste. (continua)

```

44
45 // chama os métodos getMinimum e getMaximum
46 System.out.printf("Lowest grade is %d\nHighest grade is %d\n\n",
47 getMinimum(), getMaximum());
48
49 // chama outputBarChart para imprimir o gráfico de distribuição de notas
50 outputBarChart();
51 } // fim do método processGrades
52
53 // encontra a nota mínima
54 public int getMinimum()
55 {
56 int lowGrade = grades[0]; // presume que grades[0] é a menor
57
58 // loop pelo array grades
59 for (int grade : grades)
60 {
61 // se grade é menor que lowGrade, o atribui a lowGrade
62 if (grade < lowGrade)
63 lowGrade = grade; // nova nota mais baixa
64 } // end for
65
66 return lowGrade; // retorna a nota mais baixa
67 } // fim do método getMinimum
68
69 // encontra a nota máxima
70 public int getMaximum()
71 {
72 int highGrade = grades[0]; // presume que grades[0] é a maior
73
74 // loop pelo array grades
75 for (int grade : grades)
76 {
77 // se grade é maior que highGrade, o atribui a highGrade
78 if (grade > highGrade)
79 highGrade = grade; // nova nota mais alta
80 } // fim do for
81
82 return highGrade; // retorna a nota mais alta
83 } // fim do método getMaximum
84
85 // determina a nota média do teste
86 public double getAverage()
87 {
88 int total = 0; // inicializa total
89
90 // soma as notas de um aluno
91 for (int grade : grades)
92 total += grade;
93
94 // retorna a média das notas
95 return (double) total / grades.length;
96 } // fim do método getAverage
97
98 // gera o gráfico de barras que mostra a distribuição das notas
99 public void outputBarChart()
100 {
101 System.out.println("Grade distribution:");
102
103 // armazena a frequência das notas em cada intervalo de 10 notas

```

**Figura E.13** Classe GradeBook usando um array para armazenar notas de teste. (continua)



```

104 int[] frequency = new int[11];
105
106 // para cada nota, incrementa a frequência apropriada
107 for (int grade : grades)
108 ++frequency[grade / 10];
109
110 // para cada frequência de nota, imprime uma barra no gráfico
111 for (int count = 0; count < frequency.length; count++)
112 {
113 // gera o rótulo da barra ("00-09: ", ..., "90-99: ", "100: ")
114 if (count == 10)
115 System.out.printf("%5d: ", 100);
116 else
117 System.out.printf("%02d-%02d: ",
118 count * 10, count * 10 + 9);
119
120 // imprime barra de asteriscos
121 for (int stars = 0; stars < frequency[count]; stars++)
122 System.out.print("*");
123
124 System.out.println(); // começa uma nova linha de saída
125 } // fim do for externo
126 } // fim do método outputBarChart
127
128 // gera o conteúdo do array grades
129 public void outputGrades()
130 {
131 System.out.println("The grades are:\n");
132
133 // gera a nota de cada aluno
134 for (int student = 0; student < grades.length; student++)
135 System.out.printf("Student %2d: %3d\n",
136 student + 1, grades[student]);
137 } // fim do método outputGrades
138 } // fim da classe GradeBook

```

**Figura E.13** Classe GradeBook usando um array para armazenar notas de teste.

O método `processGrades` chama em seguida o método `getAverage` (linha 43) para obter a média das notas no array. O método `getAverage` (linhas 86 a 96) usa uma instrução `for` melhorada para totalizar os valores no array `grades`, antes de calcular a média. O parâmetro no cabeçalho do `for` melhorado (por exemplo, `int grade`) indica que, para cada iteração, a variável `int grade` assume um valor no array `grades`. O cálculo da média na linha 95 usa `grades.length` para determinar o número de notas que estão sendo consideradas.

As linhas 46 a 47 no método `processGrades` chamam os métodos `getMinimum` e `getMaximum` para determinar, respectivamente, a menor e a maior nota de qualquer aluno no exame. Cada um desses métodos utiliza uma instrução `for` melhorada para fazer loop pelo array `grades`. As linhas 59 a 64 no método `getMinimum` fazem loop pelo array. As linhas 62 e 63 comparam cada nota com `lowGrade`; se uma nota é menor que `lowGrade`, `lowGrade` é configurada com essa nota. Quando a linha 66 é executada, `lowGrade` contém a menor nota no array. O método `getMaximum` (linhas 70 a 83) funciona de forma semelhante ao método `getMinimum`.

Por fim, a linha 50 no método `processGrades` chama o método `outputBarChart` para imprimir um gráfico de distribuição dos dados de nota usando uma técnica seme-

lhante à da Fig. E.5. Naquele exemplo, calculamos manualmente o número de notas em cada categoria (isto é, 0 a 9, 10 a 19, ..., 90 a 99 e 100) simplesmente examinando um conjunto de notas. Neste exemplo, as linhas 107 e 108 utilizam uma técnica semelhante à das Figs. E.6 e E.7 para calcular a frequência de notas em cada categoria. A linha 104 declara e cria o array `frequency` com 11 valores `int` para armazenar a frequência de notas em cada categoria. Para cada nota no array `grades`, as linhas 107 e 108 incrementam o elemento apropriado do array `frequency`. Para determinar qual elemento deve ser incrementado, a linha 108 divide a nota (`grade`) por 10 usando divisão inteira. Por exemplo, se `grade` é 85, a linha 108 incrementa `frequency[8]` para atualizar a contagem de notas no intervalo de 80 a 89. Em seguida, as linhas 111 a 125 imprimem o gráfico de barras (consulte a Fig. E.14) com base nos valores do array `frequency`. Assim como as linhas 23 e 24 da Fig. E.5, as linhas 121 e 122 da Fig. E.13 utilizam um valor do array `frequency` para determinar o número de asteriscos a exibir em cada barra.

### Classe *GradeBookTest* que demonstra a classe *GradeBook*

O aplicativo da Fig. E.14 cria um objeto da classe *GradeBook* (Fig. E.13) usando o array `int gradesArray` (declarado e inicializado na linha 10 da Fig. E.14). As linhas 12 e 13 passam um nome de curso e `gradesArray` para o construtor de *GradeBook*. A linha 14 exibe uma mensagem de boas-vindas e a linha 15 chama o método `processGrades` do objeto *GradeBook*. A saída resume as 10 notas em `myGradeBook`.



#### Observação sobre engenharia de software E.1

*Um sistema de teste (test harness) ou aplicativo de teste é responsável por criar um objeto da classe que está sendo testada e por fornecer dados. Esses dados podem ser provenientes de qualquer uma de várias fontes. Os dados de teste podem ser colocados diretamente em um array com um inicializador de array, podem ser digitados pelo usuário em um teclado, podem vir de um arquivo ou de uma rede. Depois de passar esses dados para o construtor da classe para instanciar o objeto, o sistema de teste deve contar com o objeto para testar seus métodos e manipular seus dados. Coletar dados no sistema de teste dessa forma permite à classe manipular dados de várias fontes.*

```

1 // Fig. E.14: GradeBookTest.java
2 // GradeBookTest cria um objeto GradeBook usando um array de notas e,
3 // então, chama o método processGrades para analisá-las.
4 public class GradeBookTest
5 {
6 // o método main inicia a execução do programa
7 public static void main(String[] args)
8 {
9 // array de notas de alunos
10 int[] gradesArray = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
11
12 GradeBook myGradeBook = new GradeBook(
13 "CS101 Introduction to Java Programming", gradesArray);
14 myGradeBook.displayMessage();
15 myGradeBook.processGrades();
16 } // fim de main
17 } // fim da classe GradeBookTest

```

**Figura E.14** *GradeBookTest* cria um objeto *GradeBook* usando um array de notas e, então, chama o método `processGrades` para analisá-las. (continua)

```
Welcome to the grade book for
CS101 Introduction to Java Programming!
```

```
The grades are:
```

```
Student 1: 87
Student 2: 68
Student 3: 94
Student 4: 100
Student 5: 83
Student 6: 78
Student 7: 85
Student 8: 91
Student 9: 76
Student 10: 87
```

```
Class average is 84.90
Lowest grade is 68
Highest grade is 100
```

```
Grade distribution:
```

```
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
100: *
```

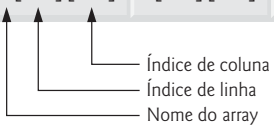
**Figura E.14** GradeBookTest cria um objeto GradeBook usando um array de notas e, então, chama o método processGrades para analisá-las.

## E.9 Arrays multidimensionais

Arrays multidimensionais com duas dimensões são frequentemente usados para representar *tabelas* de valores compostas de informações organizadas em *linhas* e *colunas*. Para identificar um elemento em particular da tabela, devemos especificar dois índices. *Por convenção*, o primeiro identifica a linha do elemento e o segundo, sua coluna. Os arrays que exigem dois índices para identificar um elemento em particular são chamados de **arrays bidimensionais**. (Os arrays multidimensionais podem ter mais de duas dimensões.) A linguagem Java não oferece suporte para arrays multidimensionais diretamente, mas permite que você especifique arrays unidimensionais cujos elementos também são arrays unidimensionais, obtendo assim o mesmo efeito. A Figura E.15 ilustra um array bidimensional chamado *a* que contém três linhas e quatro colunas (isto é, um array de três por quatro). Em geral, um array com *m* linhas e *n* colunas é chamado de array de ***m* por *n***.

Todo elemento do array *a* é identificado na Fig. E.15 por uma *expressão de acesso ao array* da forma *a[linha][coluna]*; *a* é o nome do array, *linha* e *coluna* são os índices que identificam exclusivamente cada elemento do array *a* pelo número de linha e coluna. Todos os nomes dos elementos na *linha* 0 têm o primeiro índice 0, e todos os nomes dos elementos na *coluna* 3 têm o segundo índice 3.

|         | Coluna 0    | Coluna 1    | Coluna 2    | Coluna 3    |
|---------|-------------|-------------|-------------|-------------|
| Linha 0 | a[ 0 ][ 0 ] | a[ 0 ][ 1 ] | a[ 0 ][ 2 ] | a[ 0 ][ 3 ] |
| Linha 1 | a[ 1 ][ 0 ] | a[ 1 ][ 1 ] | a[ 1 ][ 2 ] | a[ 1 ][ 3 ] |
| Linha 2 | a[ 2 ][ 0 ] | a[ 2 ][ 1 ] | a[ 2 ][ 2 ] | a[ 2 ][ 3 ] |



**Figura E.15** Array bidimensional com três linhas e quatro colunas.

### Arrays de arrays unidimensionais

Assim como os arrays unidimensionais, os arrays multidimensionais podem ser inicializados com inicializadores de array em declarações. Um array bidimensional **b** com duas linhas e duas colunas poderia ser declarado e inicializado com **inicializadores de array aninhados**, como segue:

```
int[][] b = { { 1, 2 }, { 3, 4 } };
```

Os valores iniciais são agrupados por linha em chaves. Assim, 1 e 2 inicializam `b[0][0]` e `b[0][1]`, respectivamente, e 3 e 4 inicializam `b[1][0]` e `b[1][1]`, respectivamente. O compilador conta o número de inicializadores de array aninhados (representados por conjuntos de chaves dentro de chaves externas) para determinar o número de linhas no array **b**, e conta os valores inicializadores no inicializador de array aninhado de uma linha para determinar o número de colunas nessa linha. Conforme veremos em breve, isso significa que as *linhas podem ter diferentes comprimentos*. Os arrays multidimensionais são mantidos como arrays de arrays unidimensionais. Portanto, o array **b** na declaração anterior é, na verdade, composto por dois arrays unidimensionais separados – um contendo os valores da primeira lista de inicializadores aninhada { 1, 2 } e outro contendo os valores da segunda lista de inicializadores aninhada { 3, 4 }. Assim, o próprio array **b** é um array de dois elementos, cada um sendo um array unidimensional de valores `int`.

### Arrays bidimensionais com linhas de comprimentos diferentes

O modo como os arrays multidimensionais são representados os torna muito flexíveis. De fato, os comprimentos das linhas no array **b** *não* precisam ser iguais. Por exemplo,

```
int[][] b = { { 1, 2 }, { 3, 4, 5 } };
```

cria o array inteiro **b** com dois elementos (determinado pelo número de inicializadores de array aninhados) que representam as linhas do array bidimensional. Cada elemento de **b** é uma referência para um array unidimensional de variáveis `int`. O array `int` da linha 0 é um array unidimensional com dois elementos (1 e 2), e o array `int` da linha 1 é um array unidimensional com três elementos (3, 4 e 5).

### Criando arrays bidimensionais com expressões de criação de array

Um array multidimensional com o mesmo número de colunas em cada linha pode ser criado com uma expressão de criação de array. Por exemplo, as linhas a seguir declaram o array **b** e atribuem a ele uma referência para um array de três por quatro:

```
int[][] b = new int[3][4];
```

Nesse caso, usamos os valores literais 3 e 4 para especificar o número de linhas e o número de colunas, respectivamente, mas isso não é obrigatório. Os programas também podem usar variáveis para especificar dimensões de array, pois *new cria arrays em tempo de execução – não em tempo de compilação*. Como nos arrays unidimensionais, os elementos de um array multidimensional são inicializados quando o objeto array é criado.

Um array multidimensional no qual cada linha tem um número diferente de colunas pode ser criado como segue:

```
int[][] b = new int[2][]; // cria duas linhas
b[0] = new int[5]; // cria cinco colunas para a linha 0
b[1] = new int[3]; // cria três colunas para a linha 1
```

As instruções anteriores criam um array bidimensional com duas linhas. A linha 0 tem cinco colunas e a linha 1 tem três.

### Exemplo de array bidimensional: exibindo valores de elemento

A Figura E.16 demonstra a inicialização de arrays bidimensionais com inicializadores de array e com o uso de loops for aninhados para **percorrer** os arrays (isto é, manipular todos os elementos de cada array). O método main da classe InitArray declara dois arrays. A declaração de array1 (linha 9) usa inicializadores de array aninhados de *mesmo* comprimento para inicializar a primeira linha com os valores 1, 2 e 3, e a segunda linha com os valores 4, 5 e 6. A declaração de array2 (linha 10) usa inicializadores aninhados de comprimentos *diferentes*. Nesse caso, a primeira linha é inicializada com dois elementos, com os valores 1 e 2 respectivamente. A segunda linha é inicializada com um elemento, com o valor 3. A terceira linha é inicializada com três elementos, com os valores 4, 5 e 6 respectivamente.

```
1 // Fig. E.16: InitArray.java
2 // Inicializando arrays bidimensionais.
3
4 public class InitArray
5 {
6 // cria e gera arrays bidimensionais na saída
7 public static void main(String[] args)
8 {
9 int[][] array1 = { { 1, 2, 3 }, { 4, 5, 6 } };
10 int[][] array2 = { { 1, 2 }, { 3 }, { 4, 5, 6 } };
11
12 System.out.println("Values in array1 by row are");
13 outputArray(array1); // exibe array1 por linha
14
15 System.out.println("\nValues in array2 by row are");
16 outputArray(array2); // exibe array2 por linha
17 } // fim de main
18
19 // gera linhas e colunas de um array bidimensional
20 public static void outputArray(int[][] array)
21 {
22 // faz loop pelas linhas do array
23 for (int row = 0; row < array.length; row++)
24 {
25 // faz loop pelas colunas da linha atual
26 for (int column = 0; column < array[row].length; column++)
27 System.out.printf("%d ", array[row][column]);
28 }
```

**Figura E.16** Inicializando arrays bidimensionais. (continua)

```

29 System.out.println(); // inicia nova linha de saída
30 } // fim do for externo
31 } // fim do método outputArray
32 } // fim da classe InitArray

```

```

Values in array1 by row are
1 2 3
4 5 6

Values in array2 by row are
1 2
3
4 5 6

```

**Figura E.16** Inicializando arrays bidimensionais.

As linhas 13 e 16 chamam o método `outputArray` (linhas 20 a 31) para gerar os elementos de `array1` e `array2`, respectivamente. O parâmetro do método `outputArray` – `int[][] array` – indica que o método recebe um array bidimensional. A instrução `for` (linhas 23 a 30) gera as linhas de um array bidimensional. Na condição de continuação do loop da instrução `for` externa, a expressão `array.length` determina o número de linhas no array. Na instrução `for` interna, a expressão `array[row].length` determina o número de colunas na linha atual do array. A condição da instrução `for` interna permite que o loop determine o número exato de colunas em cada linha.

### ***Manipulações comuns de arrays multidimensionais executadas com instruções for***

Muitas manipulações comuns de array utilizam instruções `for`. Como exemplo, a instrução `for` a seguir configura todos os elementos na linha 2 do array `a` da Fig. E.15 com zero:

```

for (int column = 0; column < a[2].length; column++)
 a[2][column] = 0;

```

Especificamos a linha 2; portanto, sabemos que o primeiro índice é sempre 2 (0 é a primeira linha e 1 é a segunda). Esse loop `for` varia apenas o segundo índice (isto é, o índice de coluna). Se a linha 2 do array `a` contém quatro elementos, então a instrução `for` anterior é equivalente às instruções de atribuição

```

a[2][0] = 0;
a[2][1] = 0;
a[2][2] = 0;
a[2][3] = 0;

```

A instrução `for` aninhada a seguir totaliza os valores de todos os elementos do array `a`:

```

int total = 0;

for (int row = 0; row < a.length; row++)
{
 for (int column = 0; column < a[row].length; column++)
 total += a[row][column];
} // fim do for externo

```

Essas instruções `for` aninhadas totalizam os elementos do array uma linha por vez. A instrução `for` externa começa configurando o índice de `row` como 0 para que os elementos da primeira linha possam ser totalizados pela instrução `for` interna. Então, o `for` externo

incrementa row por 1 para que a segunda linha possa ser totalizada. Em seguida, o for externo incrementa row por 2 para que a terceira linha possa ser totalizada. A variável total pode ser exibida quando a instrução for externa terminar. No próximo exemplo, mostramos como processar um array bidimensional de maneira semelhante, usando instruções for melhoradas e aninhadas.

## E.10 Estudo de caso: classe GradeBook usando um array bidimensional

Na seção E.8, apresentamos a classe GradeBook (Fig. E.13), a qual usava um array unidimensional para armazenar notas de alunos em um único exame. Na maioria dos semestres, os alunos fazem vários exames. Provavelmente, os professores querem analisar as notas de todo um semestre, tanto de um aluno como de toda a classe.

### *Armazenando notas de aluno em um array bidimensional na classe GradeBook*

A Figura E.17 contém uma classe GradeBook que usa um array bidimensional grades para armazenar as notas de vários alunos obtidas em vários exames. Cada linha do array representa as notas de um aluno para o curso inteiro, e cada coluna representa as notas de todos os alunos que fizeram um exame em particular. A classe GradeBookTest (Fig. E.18) passa o array como argumento para o construtor de GradeBook. Neste exemplo, usamos um array de 10 por 3 para as notas de 10 alunos em três exames. Cinco métodos fazem manipulações de array para processar as notas. Cada método é semelhante ao seu correspondente na versão com array unidimensional de GradeBook (Fig. E.13). O método getMinimum (linhas 52 a 70) determina a menor nota de qualquer aluno no semestre. O método getMaximum (linhas 73 a 91) determina a maior nota de qualquer aluno no semestre. O método getAverage (linhas 94 a 104) determina a média, no semestre, de um aluno em particular. O método outputBarChart (linhas 107 a 137) gera um gráfico de barras para as notas dos alunos no semestre inteiro. O método outputGrades (linhas 140 a 164) gera o array em formato tabular, junto com a média semestral de cada aluno.

```

1 // Fig. E.17: GradeBook.java
2 // Classe GradeBook usando um array bidimensional para armazenar notas.
3
4 public class GradeBook
5 {
6 private String courseName; // nome do curso dessa folha de notas
7 private int[][] grades; // array bidimensional de notas de aluno
8
9 // o construtor de dois argumentos inicializa courseName e array grades
10 public GradeBook(String name, int[][] gradesArray)
11 {
12 courseName = name; // inicializa courseName
13 grades = gradesArray; // armazena grades
14 } // fim do construtor de dois argumentos de GradeBook
15
16 // método para configurar o nome do curso
17 public void setCourseName(String name)
18 {
19 courseName = name; // armazena o nome do curso
20 } // fim do método setCourseName

```

**Figura E.17** Classe GradeBook usando um array bidimensional para armazenar notas.

```

21
22 // método para recuperar o nome do curso
23 public String getCourseName()
24 {
25 return courseName;
26 } // fim do método getCourseName
27
28 // exibe uma mensagem de boas-vindas para o usuário do aplicativo de folha de notas
29 public void displayMessage()
30 {
31 // getCourseName recebe o nome do curso
32 System.out.printf("Welcome to the grade book for\n%s!\n\n",
33 getCourseName());
34 } // fim do método displayMessage
35
36 // efetua várias operações nos dados
37 public void processGrades()
38 {
39 // gera o array grades
40 outputGrades();
41
42 // chama os métodos getMinimum e getMaximum
43 System.out.printf("\n%s %d\n%s %d\n\n",
44 "Lowest grade in the grade book is", getMinimum(),
45 "Highest grade in the grade book is", getMaximum());
46
47 // gera gráfico de distribuição de todas as notas em todos os testes
48 outputBarChart();
49 } // fim do método processGrades
50
51 // encontra a nota mínima
52 public int getMinimum()
53 {
54 // presume que o primeiro elemento do array grades é o menor
55 int lowGrade = grades[0][0];
56
57 // faz loop pelas linhas do array grades
58 for (int[] studentGrades : grades)
59 {
60 // faz loop pelas colunas da linha atual
61 for (int grade : studentGrades)
62 {
63 // se grade é menor que lowGrade, o atribui a lowGrade
64 if (grade < lowGrade)
65 lowGrade = grade;
66 } // fim do for interno
67 } // fim do for externo
68
69 return lowGrade; // retorna a nota mais baixa
70 } // fim do método getMinimum
71
72 // encontra a nota máxima
73 public int getMaximum()
74 {
75 // presume que o primeiro elemento do array grades é o maior
76 int highGrade = grades[0][0];
77
78 // faz loop pelas linhas do array grades
79 for (int[] studentGrades : grades)
80 {
81 // faz loop pelas colunas da linha atual
82 for (int grade : studentGrades)

```

**Figura E.17** Classe GradeBook usando um array bidimensional para armazenar notas. (continua)



```

83 {
84 // se grade é maior que highGrade, o atribui a highGrade
85 if (grade > highGrade)
86 highGrade = grade;
87 } // fim do for interno
88 } // fim do for externo
89
90 return highGrade; // retorna a nota mais alta
91 } // fim do método getMaximum
92
93 // determina a nota média para um conjunto de notas em particular
94 public double getAverage(int[] setOfGrades)
95 {
96 int total = 0; // inicializa total
97
98 // soma as notas de um aluno
99 for (int grade : setOfGrades)
100 total += grade;
101
102 // retorna a média das notas
103 return (double) total / setOfGrades.length;
104 } // fim do método getAverage
105
106 // gera o gráfico de barras que mostra a distribuição geral das notas
107 public void outputBarChart()
108 {
109 System.out.println("Overall grade distribution:");
110
111 // armazena a frequência das notas em cada intervalo de 10 notas
112 int[] frequency = new int[11];
113
114 // para cada nota em GradeBook, incrementa a frequência apropriada
115 for (int[] studentGrades : grades)
116 {
117 for (int grade : studentGrades)
118 ++frequency[grade / 10];
119 } // fim do for externo
120
121 // para cada frequência de nota, imprime uma barra no gráfico
122 for (int count = 0; count < frequency.length; count++)
123 {
124 // gera o rótulo da barra ("00-09: ", ..., "90-99: ", "100: ")
125 if (count == 10)
126 System.out.printf("%5d: ", 100);
127 else
128 System.out.printf("%02d-%02d: ",
129 count * 10, count * 10 + 9);
130
131 // imprime barra de asteriscos
132 for (int stars = 0; stars < frequency[count]; stars++)
133 System.out.print("*");
134
135 System.out.println(); // começa uma nova linha de saída
136 } // fim do for externo
137 } // fim do método outputBarChart
138
139 // gera o conteúdo do array grades
140 public void outputGrades()
141 {
142 System.out.println("The grades are:\n");
143 System.out.print(" "); // alinha os cabeçalhos de coluna
144

```

**Figura E.17** Classe GradeBook usando um array bidimensional para armazenar notas.

```

145 // cria um cabeçalho de coluna para cada um dos testes
146 for (int test = 0; test < grades[0].length; test++)
147 System.out.printf("Test %d ", test + 1);
148
149 System.out.println("Average"); // cabeçalho de coluna para média do aluno
150
151 // cria linhas/colunas de texto representando as notas do array
152 for (int student = 0; student < grades.length; student++)
153 {
154 System.out.printf("Student %2d", student + 1);
155
156 for (int test : grades[student]) // gera as notas do aluno
157 System.out.printf("%8d", test);
158
159 // chama o método getAverage para calcular a nota média do aluno
160 // passa linha de notas como argumento para getAverage
161 double average = getAverage(grades[student]);
162 System.out.printf("%9.2f\n", average);
163 } // fim do for externo
164 } // fim do método outputGrades
165 } // fim da classe GradeBook

```

**Figura E.17** Classe GradeBook usando um array bidimensional para armazenar notas.

### **Métodos *getMinimum* e *getMaximum***

Os métodos *getMinimum*, *getMaximum*, *outputBarChart* e *outputGrades* fazem loop pelo array *grades* usando instruções *for* aninhadas – por exemplo, a instrução *for* melhorada e aninhada da declaração do método *getMinimum* (linhas 58 a 67). A instrução *for* melhorada externa itera pelo array bidimensional *grades*, atribuindo sucessivas linhas ao parâmetro *studentGrades* em sucessivas iterações. Os colchetes após o nome do parâmetro indicam que *studentGrades* se refere a um array *int* unidimensional – ou seja, uma linha no array *grades* contendo as notas de um aluno. Para encontrar a menor nota no geral, a instrução *for* interna compara os elementos do array unidimensional *studentGrades* atual com a variável *lowGrade*. Por exemplo, na primeira iteração do *for* externo, a linha 0 de *grades* é atribuída ao parâmetro *studentGrades*. Então, a instrução *for* melhorada interna faz loop por *studentGrades* e compara cada valor de *grade* com *lowGrade*. Se uma nota é menor que *lowGrade*, *lowGrade* é configurada com essa nota. Na segunda iteração da instrução *for* melhorada externa, a linha 1 de *grades* é atribuída a *studentGrades* e os elementos dessa linha são comparados com a variável *lowGrade*. Isso se repete até que todas as linhas de *grades* tenham sido percorridas. Quando a execução da instrução aninhada termina, *lowGrade* contém a menor nota do array bidimensional. O método *getMaximum* funciona de modo semelhante ao método *getMinimum*.

### **O método *outputBarChart***

O método *outputBarChart* (linhas 107 a 137) é praticamente idêntico ao da Fig. E.13. Contudo, para gerar a distribuição geral de notas para um semestre inteiro, aqui o método usa instruções *for* melhoradas e aninhadas (linhas 115 a 119) para criar o array unidimensional *frequency* com base em todas as notas do array bidimensional. O restante do código em cada um dos dois métodos *outputBarChart* que exibem o gráfico é idêntico.

### **O método *outputGrades***

O método *outputGrades* (linhas 140 a 164) usa instruções *for* aninhadas para gerar os valores do array *grades* e a média semestral de cada aluno. A saída (Fig. E.18) mostra o

resultado, que é semelhante ao formato tabular de uma folha de notas real utilizada por um professor. As linhas 146 a 147 imprimem os cabeçalhos de coluna para cada teste. Usamos uma instrução `for` controlada por contador aqui para podermos identificar cada teste com um número. Do mesmo modo, a instrução `for` nas linhas 152 a 163 primeiramente gera um rótulo de linha usando uma variável contadora para identificar cada aluno (linha 154). Embora os índices `array` comecem em 0, as linhas 147 e 154 geram `test + 1` e `student + 1`, respectivamente, para produzir números de teste e aluno a partir de 1 (consulte a Fig. E.18). A instrução `for` interna (linhas 156 e 157) usa a variável contadora `student` da instrução `for` externa para fazer loop por uma linha específica do array `grades` e gerar a nota no teste de cada aluno. Uma instrução `for` melhorada pode ser aninhada em uma instrução `for` controlada por contador e vice-versa. Por fim, a linha 161 obtém a média semestral de cada aluno, passando a linha atual de `grades` (isto é, `grades[student]`) para o método `getAverage`.

### O método `getAverage`

O método `getAverage` (linhas 94 a 104) recebe um único argumento – um array unidimensional de resultados de teste para um aluno em particular. Quando a linha 161 chama `getAverage`, o argumento é `grades[student]`, o qual especifica que uma linha em particular do array bidimensional `grades` deve ser passada para `getAverage`. Por exemplo, com base no array criado na Fig. E.18, o argumento `grades[1]` representa os três valores (um array unidimensional de notas) armazenados na linha 1 do array bidimensional `grades`. Lembre-se de que um array bidimensional é aquele cujos elementos são arrays unidimensionais. O método `getAverage` calcula a soma dos elementos do array, divide o total pelo número de resultados de teste e retorna o resultado de ponto flutuante como um valor `double` (linha 103).

### Classe `GradeBookTest` que demonstra a classe `GradeBook`

A Figura E.18 cria um objeto da classe `GradeBook` (Fig. E.17) usando o array bidimensional de valores `int` chamado `gradesArray` (declarado e inicializado nas linhas 10 a 19). As linhas 21 e 22 passam um nome de curso e `gradesArray` para o construtor de `GradeBook`. Então, as linhas 23 e 24 chamam os métodos `displayMessage` e `processGrades` de `myGradeBook` para exibir uma mensagem de boas-vindas e obter um relatório resumindo as notas do aluno para o semestre, respectivamente.

```

1 // Fig. E.18: GradeBookTest.java
2 // GradeBookTest cria objeto GradeBook usando um array bidimensional
3 // de notas e, então, chama o método processGrades para analisá-las.
4 public class GradeBookTest
5 {
6 // o método main inicia a execução do programa
7 public static void main(String[] args)
8 {
9 // array bidimensional de notas de aluno
10 int[][] gradesArray = { { 87, 96, 70 },
11 { 68, 87, 90 },
12 { 94, 100, 90 },
13 { 100, 81, 82 },
14 { 83, 65, 85 },
15 { 78, 87, 65 },

```

**Figura E.18** `GradeBookTest` cria objeto `GradeBook` usando um array bidimensional de notas e, então, chama o método `processGrades` para analisá-las.

```

16 { 85, 75, 83 },
17 { 91, 94, 100 },
18 { 76, 72, 84 },
19 { 87, 93, 73 } };
20
21 GradeBook myGradeBook = new GradeBook(
22 "CS101 Introduction to Java Programming", gradesArray);
23 myGradeBook.displayMessage();
24 myGradeBook.processGrades();
25 } // fim de main
26 } // fim da classe GradeBookTest

```

```

Welcome to the grade book for
CS101 Introduction to Java Programming!

```

The grades are:

|            | Test 1 | Test 2 | Test 3 | Average |
|------------|--------|--------|--------|---------|
| Student 1  | 87     | 96     | 70     | 84.33   |
| Student 2  | 68     | 87     | 90     | 81.67   |
| Student 3  | 94     | 100    | 90     | 94.67   |
| Student 4  | 100    | 81     | 82     | 87.67   |
| Student 5  | 83     | 65     | 85     | 77.67   |
| Student 6  | 78     | 87     | 65     | 76.67   |
| Student 7  | 85     | 75     | 83     | 81.00   |
| Student 8  | 91     | 94     | 100    | 95.00   |
| Student 9  | 76     | 72     | 84     | 77.33   |
| Student 10 | 87     | 93     | 73     | 84.33   |

```

Lowest grade in the grade book is 65
Highest grade in the grade book is 100

```

```

Overall grade distribution:
00-09:
10-19:
20-29:
30-39:

```

**Figura E.18** GradeBookTest cria objeto GradeBook usando um array bidimensional de notas e, então, chama o método processGrades para analisá-las.

## E.11 Classe Arrays

A classe **Arrays** evita que você reinvente a roda, fornecendo métodos estáticos para manipulações comuns de array. Esses métodos incluem **sort** para classificar um array (isto é, organizar os elementos em ordem crescente), **binarySearch** para pesquisar um array (isto é, determinar se um array contém um valor específico e, se assim for, onde ele está localizado), **equals** para comparar arrays e **fill** para colocar valores em um array. Eles são sobrecarregados para arrays de tipo primitivo e para arrays de objetos. Nosso enfoque nesta seção é o uso dos recursos predefinidos fornecidos pela API Java.

A Figura E.19 usa os métodos **sort**, **binarySearch**, **equals** e **fill** de **Arrays**, e mostra como copiar arrays com o método estático **arraycopy** da classe **System**. Em **main**, a linha 11 classifica os elementos do array **doubleArray**. O método estático **sort** da classe **Arrays** classifica os elementos do array em ordem *crescente*, por padrão. As versões sobrecarregadas de **sort** permitem classificar um intervalo específico de elementos. As linhas 12 a 15 geram o array ordenado.

```

1 // Fig. E.19: ArrayManipulations.java
2 // Métodos da classe Arrays e System.arraycopy.
3 import java.util.Arrays;
4
5 public class ArrayManipulations
6 {
7 public static void main(String[] args)
8 {
9 // classifica doubleArray em ordem crescente
10 double[] doubleArray = { 8.4, 9.3, 0.2, 7.9, 3.4 };
11 Arrays.sort(doubleArray);
12 System.out.printf("\ndoubleArray: ");
13
14 for (double value : doubleArray)
15 System.out.printf("%.1f ", value);
16
17 // preenche array de 10 elementos com valores 7
18 int[] filledIntArray = new int[10];
19 Arrays.fill(filledIntArray, 7);
20 displayArray(filledIntArray, "filledIntArray");
21
22 // copia o array intArray no array intArrayCopy
23 int[] intArray = { 1, 2, 3, 4, 5, 6 };
24 int[] intArrayCopy = new int[intArray.length];
25 System.arraycopy(intArray, 0, intArrayCopy, 0, intArray.length);
26 displayArray(intArray, "intArray");
27 displayArray(intArrayCopy, "intArrayCopy");
28
29 // compara intArray e intArrayCopy quanto à igualdade
30 boolean b = Arrays.equals(intArray, intArrayCopy);
31 System.out.printf("\n\nintArray %s intArrayCopy\n",
32 (b ? "==" : "!="));
33
34 // compara intArray e filledIntArray quanto à igualdade
35 b = Arrays.equals(intArray, filledIntArray);
36 System.out.printf("intArray %s filledIntArray\n",
37 (b ? "==" : "!="));
38
39 // procura o valor 5 em intArray
40 int location = Arrays.binarySearch(intArray, 5);
41
42 if (location >= 0)
43 System.out.printf(
44 "Found 5 at element %d in intArray\n", location);
45 else
46 System.out.println("5 not found in intArray");
47
48 // procura o valor 8763 em intArray
49 location = Arrays.binarySearch(intArray, 8763);
50
51 if (location >= 0)
52 System.out.printf(
53 "Found 8763 at element %d in intArray\n", location);
54 else
55 System.out.println("8763 not found in intArray");
56 } // fim de main
57
58 // gera os valores em cada array
59 public static void displayArray(int[] array, String description)
60 {
61 System.out.printf("\n%s: ", description);
62 }

```

**Figura E.19** Métodos da classe Arrays e System.arraycopy. (continua)

```

63 for (int value : array)
64 System.out.printf("%d ", value);
65 } // fim do método displayArray
66 } // fim da classe ArrayManipulations

```

```

doubleArray: 0.2 3.4 7.9 8.4 9.3
filledIntArray: 7 7 7 7 7 7 7 7 7
intArray: 1 2 3 4 5 6
intArrayCopy: 1 2 3 4 5 6
intArray == intArrayCopy
intArray != filledIntArray
Found 5 at element 4 in intArray
8763 not found in intArray

```

**Figura E.19** Métodos da classe Arrays e System.arraycopy.

A linha 19 chama o método estático `fill` da classe `Arrays` para preencher todos os 10 elementos de `filledIntArray` com 7. As versões sobrecarregadas de `fill` permitem preencher um intervalo específico de elementos com o mesmo valor. A linha 20 chama o método `displayArray` de nossa classe (declarado nas linhas 59 a 65) para gerar o conteúdo de `filledIntArray`.

A linha 25 copia os elementos de `intArray` em `intArrayCopy`. O primeiro argumento (`intArray`) passado para o método `arraycopy` de `System` é o array a partir do qual os elementos devem ser copiados. O segundo argumento (0) é o índice que especifica o ponto de partida no intervalo de elementos a copiar do array. Esse valor pode ser qualquer índice de array válido. O terceiro argumento (`intArrayCopy`) especifica o array de destino que vai armazenar a cópia. O quarto argumento (0) especifica o índice no array de destino onde o primeiro elemento copiado deve ser armazenado. O último argumento especifica o número de elementos a copiar do array do primeiro argumento. Neste caso, copiamos todos os elementos do array.

As linhas 30 e 35 chamam o método estático `equals` da classe `Arrays` para determinar se todos os elementos de dois arrays são equivalentes. Se os arrays contêm os mesmos elementos na mesma ordem, o método retorna `true`; caso contrário, retorna `false`.

As linhas 40 e 49 chamam o método estático `binarySearch` da classe `Arrays` para fazer uma busca binária em `intArray`, usando o segundo argumento (5 e 8763, respectivamente) como chave. Se o valor for encontrado, `binarySearch` retorna o índice do elemento; caso contrário, `binarySearch` retorna um valor negativo. O valor negativo retornado é baseado no ponto de inserção da chave de busca – o índice onde a chave seria inserida no array se estivéssemos efetuando uma operação de inserção. Depois de determinar o ponto de inserção, `binarySearch` muda seu sinal para negativo e subtrai 1 para obter o valor de retorno. Por exemplo, na Fig. E.19, o ponto de inserção para o valor 8763 é o elemento com índice 6 no array. O método `binarySearch` muda o ponto de inserção para -6, subtrai 1 dele e retorna o valor -7. Subtrair 1 do ponto de inserção garante que o método `binarySearch` retorne valores positivos ( $\geq 0$ ), se e somente se a chave for encontrada. Esse valor de retorno é útil para inserir elementos em um array ordenado.



### Erro de programação comum E.3

*Passar um array não ordenado para `binarySearch` é um erro de lógica – o valor retornado é indefinido.*

## E.12 Introdução às coleções e à classe ArrayList

A API Java fornece várias estruturas de dados predefinidas, denominadas **coleções**, usadas para armazenar grupos de objetos relacionados. Essas classes fornecem métodos eficientes que organizam, armazenam e recuperam seus dados sem exigir que se saiba como os dados estão sendo armazenados. Isso reduz o tempo de desenvolvimento de aplicativos.

Você usou arrays para armazenar sequências de objetos. Os arrays não mudam de tamanho automaticamente em tempo de execução a fim de acomodar mais elementos. A classe de coleção **ArrayList<T>** (do pacote `java.util`) fornece uma solução conveniente para esse problema – ela pode mudar seu tamanho *dinamicamente* para acomodar mais elementos. O **T** (por convenção) é um *espaço reservado* – ao declarar um novo `ArrayList`, substitua-o pelo tipo de elementos que `ArrayList` deve armazenar. Isso é semelhante a especificar o tipo ao declarar um array, exceto que *somente tipos não primitivos podem ser usados com essas classes de coleção*. Por exemplo,

```
ArrayList< String > list;
```

declara `list` como uma coleção de `ArrayList` que pode armazenar somente `Strings`. As classes com essa forma de espaço reservado, que pode ser usado com qualquer tipo, são denominadas **classes genéricas**. Mais classes de coleção genéricas e genéricos são discutidos no Apêndice J. A Figura E.20 mostra alguns métodos comuns da classe `ArrayList<T>`.

| Método                  | Descrição                                                                                                                          |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <code>add</code>        | Adiciona um elemento no fim da <code>ArrayList</code> .                                                                            |
| <code>clear</code>      | Remove todos os elementos da <code>ArrayList</code> .                                                                              |
| <code>contains</code>   | Retorna <code>true</code> se a <code>ArrayList</code> contém o elemento especificado; caso contrário, retorna <code>false</code> . |
| <code>get</code>        | Retorna o elemento no índice especificado.                                                                                         |
| <code>indexOf</code>    | Retorna o índice da primeira ocorrência do elemento especificado na <code>ArrayList</code> .                                       |
| <code>remove</code>     | Sobrecarregado. Remove a primeira ocorrência do valor especificado ou do elemento no índice especificado.                          |
| <code>size</code>       | Retorna o número de elementos armazenados na <code>ArrayList</code> .                                                              |
| <code>trimToSize</code> | Reduz a capacidade da <code>ArrayList</code> ao número atual de elementos.                                                         |

**Figura E.20** Alguns métodos e propriedades da classe `ArrayList<T>`.

A Figura E.21 demonstra alguns recursos comuns de `ArrayList`. A linha 10 cria uma nova `ArrayList` de `Strings` vazia, com uma capacidade inicial padrão de 10 elementos. A capacidade indica quantos itens a `ArrayList` pode armazenar sem crescer. `ArrayList` é implementada usando um array nos bastidores. Quando a `ArrayList` cresce, precisa criar um array interno maior e copiar cada elemento no novo array. Essa é uma operação demorada. Seria ineficiente a `ArrayList` crescer sempre que um elemento é adicionado. Em vez disso, ela cresce somente quando um elemento é adicionado e o número de elementos é igual à capacidade – isto é, quando não há espaço para o novo elemento.

```

1 // Fig. E.21: ArrayListCollection.java
2 // Demonstração da coleção Genérica ArrayList<T>.
3 import java.util.ArrayList;
4
5 public class ArrayListCollection
6 {
7 public static void main(String[] args)
8 {
9 // cria uma nova ArrayList de Strings com capacidade inicial de 10
10 ArrayList< String > items = new ArrayList< String >();
11
12 items.add("red"); // anexa um item à lista
13 items.add(0, "yellow"); // insere o valor no índice 0
14
15 // cabeçalho
16 System.out.print(
17 "Display list contents with counter-controlled loop:");
18
19 // exibe as cores da lista
20 for (int i = 0; i < items.size(); i++)
21 System.out.printf(" %s", items.get(i));
22
23 // exibe as cores usando foreach no método display
24 display(items,
25 "\nDisplay list contents with enhanced for statement:");
26
27 items.add("green"); // adiciona "green" ao final da lista
28 items.add("yellow"); // adiciona "yellow" ao final da lista
29 display(items, "List with two new elements:");
30
31 items.remove("yellow"); // remove o primeiro "yellow"
32 display(items, "Remove first instance of yellow:");
33
34 items.remove(1); // remove o item no índice 1
35 display(items, "Remove second list element (green):");
36
37 // verifica se um valor está na lista
38 System.out.printf("\"red\" is %sin the list\n",
39 items.contains("red") ? "" : "not ");
40
41 // exibe o número de elementos na lista
42 System.out.printf("Size: %s\n", items.size());
43 } // fim de main
44
45 // exibe os elementos da ArrayList no console
46 public static void display(ArrayList< String > items, String header)
47 {
48 System.out.print(header); // exibe o cabeçalho
49
50 // exibe cada elemento em items
51 for (String item : items)
52 System.out.printf(" %s", item);
53
54 System.out.println(); // exibe o fim da linha
55 } // fim do método display
56 } // fim da classe ArrayListCollection

```

**Figura E.21** Demonstração da coleção genérica ArrayList<T>. (continua)



```

Display list contents with counter-controlled loop: yellow red
Display list contents with enhanced for statement: yellow red
List with two new elements: yellow red green yellow
Remove first instance of yellow: red green yellow
Remove second list element (green): red yellow
"red" is in the list
Size: 2

```

**Figura E.21** Demonstração da coleção genérica `ArrayList<T>`.

O método **add** adiciona elementos a `ArrayList` (linhas 12 e 13). O método **add** com *um* argumento anexa seu argumento no final da `ArrayList`. O método **add** com *dois* argumentos insere um novo elemento na posição especificada. O primeiro argumento é um índice. Como nos arrays, os índices de coleção começam em zero. O segundo argumento é o valor a inserir nesse índice. Os índices de todos os elementos subsequentes são incrementador por um. Normalmente, inserir um elemento é mais lento do que adicionar um elemento no final da `ArrayList`.

As linhas 20 e 21 exibem os itens da `ArrayList`. O método **size** retorna o número de elementos existentes na `ArrayList` no momento. O método **get** de `ArrayLists` (linha 21) obtém o elemento em um índice especificado. As linhas 24 e 25 exibem os elementos novamente, chamando o método **display** (definido nas linhas 46 a 55). As linhas 27 e 28 adicionam mais dois elementos à `ArrayList` e, então, a linha 29 exibe os elementos outra vez para confirmar que os dois foram adicionados no final da coleção.

O método **remove** é usado para remover um elemento com um valor específico (linha 31). Ele remove apenas o primeiro elemento especificado. Se esse elemento não está na `ArrayList`, remove nada faz. Uma versão sobrecarregada do método **remove** o elemento do índice especificado (linha 34). Quando um elemento é removido, os índices de todos os elementos após o que foi removido diminuem por um.

A linha 39 usa o método **contains** para verificar se um item está na `ArrayList`. O método **contains** retorna **true** se o elemento é encontrado na `ArrayList` e **false**, caso contrário. O método compara seu argumento com cada elemento da `ArrayList`, em ordem, de modo que pode ser ineficiente usar **contains** em uma `ArrayList` grande. A linha 42 mostra o tamanho da `ArrayList`.

## E.13 Para finalizar

Este apêndice iniciou nossa introdução às estruturas de dados explorando o uso de arrays para armazenar e recuperar dados de listas e tabelas de valores. Os exemplos do apêndice demonstraram como declarar um array, como inicializar um array e como fazer referência a elementos individuais de um array. O apêndice apresentou a instrução for melhorada para iterar por arrays. Usamos tratamento de exceção para testar exceções `ArrayIndexOutOfBoundsException`, que ocorrem quando um programa tenta acessar um elemento fora dos limites de um array. Ilustramos também como passar arrays para métodos e como declarar e manipular arrays multidimensionais.

Apresentamos a coleção genérica `ArrayList<T>`, a qual fornece toda a funcionalidade e desempenho dos arrays, junto com outros recursos úteis, como o redimensionamento dinâmico. Utilizamos os métodos **add** para adicionar novos itens no final de uma `ArrayList` e para inserir itens em uma `ArrayList`. O método **remove** foi usado para

remover a primeira ocorrência de um item especificado, e uma versão sobrecarregada de `remove` foi usada para remover um item em um índice especificado. Usamos o método `size` para obter o número de itens na `ArrayList`.

Continuamos nossa abordagem das estruturas de dados no Apêndice J. O Apêndice J apresenta a Java Collections Framework, a qual utiliza genéricos para permitir que você especifique os tipos exatos de objetos que uma estrutura de dados em particular vai armazenar. Apresenta também outras estruturas de dados predefinidas do Java. A API Collections fornece a classe `Arrays`, a qual contém métodos utilitários para manipulação de array. O Apêndice J usa vários métodos estáticos da classe `Arrays` para fazer tais manipulações, como classificar e procurar dados em um array.

Agora já apresentamos os conceitos básicos de classes, objetos, instruções de controle, métodos, arrays e coleções. No Apêndice F, examinaremos as classes e os objetos com mais profundidade.

---

## Exercícios de revisão

**E.1** Preencha os espaços em branco em cada um dos seguintes enunciados:

- Listas e tabelas de valores podem ser armazenados em \_\_\_\_\_.
- Um array é um grupo de \_\_\_\_\_ (chamadas de elementos ou componentes) contendo valores, todos do mesmo \_\_\_\_\_.
- A \_\_\_\_\_ permite iterar pelos elementos de um array sem usar um contador.
- O número usado para fazer referência a um elemento em particular do array é chamado de \_\_\_\_\_ do elemento.
- Um array que utiliza dois índices é referido como array \_\_\_\_\_.
- Use a instrução `for` melhorada \_\_\_\_\_ para percorrer números de array `double`.
- Os argumentos de linha de comando são armazenados em \_\_\_\_\_.

**E.2** Determine se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.

- Um array pode armazenar muitos tipos de valores diferentes.
- O índice de um array normalmente deve ser de tipo `float`.
- Um elemento individual de um array que é passado para um método e modificado nesse método conterá o valor modificado quando o método chamado completar a execução.

**E.3** Execute as tarefas a seguir para um array chamado `fractions`:

- Declare uma constante `ARRAY_SIZE` que seja inicializada com 10.
- Declare um array com elementos `ARRAY_SIZE` de tipo `double` e inicialize os elementos com 0.
- Faça referência ao elemento 4 do array.
- Atribua o valor 1.667 ao elemento 9 do array.
- Atribua o valor 3.333 ao elemento 6 do array.
- Some todos os elementos do array usando uma instrução `for`. Declare a variável inteira `x` como uma variável de controle para o loop.

**E.4** Execute as tarefas a seguir para um array chamado `table`:

- Declare e crie o array como um array inteiro com três linhas e três colunas. Suponha que a constante `ARRAY_SIZE` tenha sido declarada como 3.
- Quantos elementos o array contém?

- c) Use uma instrução `for` para inicializar cada elemento do array com a soma de seus índices. Suponha que as variáveis inteiras `x` e `y` sejam declaradas como variáveis de controle.

**E.5** Encontre e corrija o erro em cada um dos seguintes trechos de programa.

- a) `final int ARRAY_SIZE = 5;`  
`ARRAY_SIZE = 10;`
- b) `Assume int[] b = new int[ 10 ];`  
`for ( int i = 0; i <= b.length; i++ )`  
`b[ i ] = 1;`
- c) `Assume int[][] a = { { 1, 2 }, { 3, 4 } };`  
`a[ 1, 1 ] = 5;`

## Respostas dos exercícios de revisão

- E.1** a) arrays. b) variáveis, tipo. c) instrução `for` melhorada. d) índice (ou subscrito ou número da posição). e) bidimensional. f) `for ( double d : numbers )`. g) um array de Strings, chamado `args` por convenção.
- E.2** a) Falsa. Um array só pode armazenar valores do mesmo tipo. b) Falsa. Um índice de array deve ser um valor inteiro ou uma expressão inteira. c) Para elementos individuais de tipo primitivo de um array: falsa. Um método chamado recebe e manipula uma cópia do valor de tal elemento; portanto, as modificações não afetam o valor original. Contudo, se a referência de um array é passada para um método, as modificações nos elementos do array feitas no método chamado são, de fato, refletidas no original. Para elementos individuais de um tipo de referência: verdadeira. Um método chamado recebe uma cópia da referência desse elemento e as alterações no objeto referenciado serão refletidas no elemento do array original.
- E.3** a) `final int ARRAY_SIZE = 10;`  
b) `double[] fractions = new double[ ARRAY_SIZE ];`  
c) `fractions[ 4 ]`  
d) `fractions[ 9 ] = 1.667;`  
e) `fractions[ 6 ] = 3.333;`  
f) `double total = 0.0;`  
`for ( int x = 0; x < fractions.length; x++ )`  
`total += fractions[ x ];`
- E.4** a) `int[][] table = new int[ ARRAY_SIZE ][ ARRAY_SIZE ];`  
b) Nove.  
c) `for ( int x = 0; x < table.length; x++ )`  
`for ( int y = 0; y < table[ x ].length; y++ )`  
`table[ x ][ y ] = x + y;`
- E.5** a) Erro: atribuição de um valor a uma constante após ela ter sido inicializada.  
Correção: atribuir o valor correto à constante em uma declaração `final int ARRAY_SIZE` ou declarar outra variável.
- b) Erro: referência a um elemento de array fora dos limites do array (`b[10]`).  
Correção: mudar o operador `<=` para `<`.
- c) Erro: a indexação do array é feita incorretamente.  
Correção: mudar a instrução para `a[ 1 ][ 1 ] = 5;`.

## Exercícios

**E.6** Preencha os espaços em branco em cada um dos seguintes enunciados:

- O array unidimensional `p` contém quatro elementos. Os nomes desses elementos são \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_ e \_\_\_\_\_.
- Dar um nome a um array, declarar seu tipo e especificar o número de dimensões no array é chamado de \_\_\_\_\_ o array.
- Em um array bidimensional, o primeiro índice identifica o \_\_\_\_\_ de um elemento, e o segundo índice identifica o \_\_\_\_\_ de um elemento.
- Um array de  $m$  por  $n$  contém \_\_\_\_\_ linhas, \_\_\_\_\_ colunas e \_\_\_\_\_ elementos.
- O nome do elemento na linha 3 e coluna 5 do array `d` é \_\_\_\_\_.

**E.7** Determine se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.

- Para fazer referência a um local ou elemento em particular dentro de um array, especificamos o nome do array e o valor do elemento em particular.
- A declaração de um array reserva espaço para o array.
- Para indicar que 100 lugares devem ser reservados para o array inteiro `p`, você escreve a declaração

```
p[100];
```

- Um aplicativo que inicializa os elementos de um array de 15 elementos com zero deve conter pelo menos uma instrução `for`.
- Um aplicativo que totaliza os elementos de um array bidimensional deve conter instruções `for` aninhadas.

**E.8** Considere um array inteiro `t` de dois por três.

- Escreva uma instrução que declare e crie `t`.
- Quantas linhas `t` tem?
- Quantas colunas `t` tem?
- Quantos elementos `t` tem?
- Escreva expressões de acesso para todos os elementos da linha 1 de `t`.
- Escreva expressões de acesso para todos os elementos da coluna 2 de `t`.
- Escreva uma única instrução que configure como zero o elemento de `t` na linha 0 e coluna 1.
- Escreva instruções individuais para inicializar cada elemento de `t` com zero.
- Escreva uma instrução `for` aninhada que inicialize cada elemento de `t` com zero.
- Escreva uma instrução `for` aninhada que insira os valores dados pelo usuário para os elementos de `t`.
- Escreva uma série de instruções que determinem e exibam o menor valor em `t`.
- Escreva uma única instrução `printf` que exiba os elementos da primeira linha de `t`.
- Escreva uma instrução que totalize os elementos da terceira coluna de `t`. Não use repetição.
- Escreva uma série de instruções que exibam o conteúdo de `t` em formato tabular. Liste os índices de coluna como cabeçalhos na parte superior e os índices de linha à esquerda de cada linha.

**E.9 (Eliminação de duplicatas)** Use um array unidimensional para resolver o seguinte problema: escreva um aplicativo que insira cinco números, cada um entre 10 e 100, inclusive. À medida que cada número for lido, exiba-o apenas se não for uma duplicata de um número já lido. Forneça o “pior caso”, no qual todos os cinco números são diferentes. Use o menor array possível para resolver esse problema. Exiba o conjunto completo de valores únicos inseridos depois que o usuário digitar cada novo valor.

**E.10** Rotule os elementos do array bidimensional `sales` de três por cinco para indicar a ordem em que eles são configurados como zero pelo trecho de programa a seguir:

```
for (int row = 0; row < sales.length; row++)
{
 for (int col = 0; col < sales[row].length; col++)
 {
 sales[row][col] = 0;
 }
}
```

**E.11 (Crivo de Eratóstenes)** Um número primo é qualquer inteiro maior que 1 divisível somente por ele mesmo e por 1. O Crivo de Eratóstenes é um método para encontrar números primos. Ele funciona como segue:

- Crie um array de tipo primitivo `boolean` com todos os elementos inicializados com `true`. Os elementos do array com índices primos permanecerão `true`. Todos os outros elementos serão finalmente configurados como `false`.
- Começando com o índice de array 2, determine se um elemento dado é `true`. Se for, faça um loop pelo restante do array e configure como `false` todo elemento cujo índice for múltiplo do índice do elemento com valor `true`. Então, continue o processo com o próximo elemento com valor `true`. Para o índice de array 2, todos os elementos, depois do elemento 2 no array, que têm índices que são múltiplos de 2 (índices 4, 6, 8, 10 etc.) serão configurados como `false`; para o índice de array 3, todos os elementos, depois do elemento 3 no array, que têm índices que são múltiplos de 3 (índices 6, 9, 12, 15 etc.) serão configurados como `false`; e assim por diante. Quando esse processo termina, os elementos do array que ainda são `true` indicam que o índice é um número primo. Esses índices podem ser exibidos. Escreva um aplicativo que utilize um array de 1000 elementos para determinar e exibir os números primos entre 2 e 999. Ignore os elementos 0 e 1 do array.

**E.12 (Série de Fibonacci)** A série de Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

começa com os termos 0 e 1 e tem a característica de que cada termo sucessivo é a soma dos dois termos anteriores.

- Escreva um método `fibonacci( n )` que calcule o  $n$ -ésimo número de Fibonacci. Incorpore esse método em um aplicativo que permita ao usuário digitar o valor de  $n$ .
- Determine o maior número de Fibonacci que pode ser exibido em seu sistema.
- Modifique o aplicativo que você escreveu na parte (a) para usar `double` em vez de `int` a fim de calcular e retornar os números de Fibonacci, e use esse aplicativo modificado para repetir a parte (b).

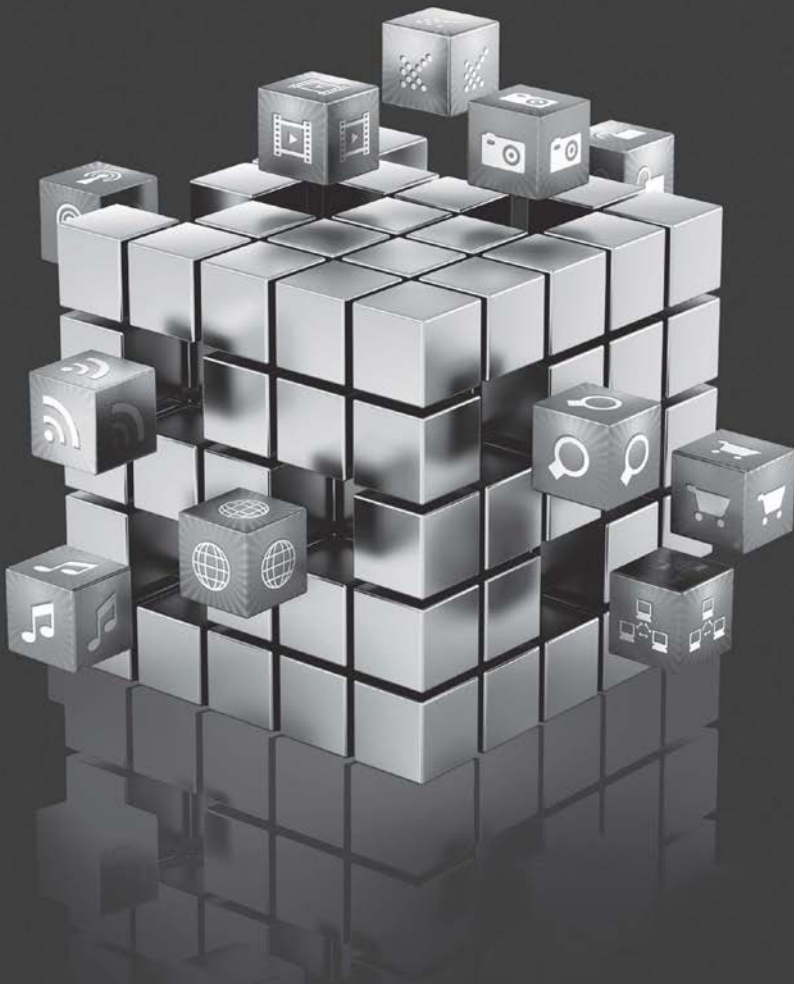
# Classes e objetos: uma investigação mais aprofundada

# F

## Objetivos

Neste capítulo, você vai:

- Aprender sobre encapsulamento e ocultação de dados.
- Usar a palavra-chave `this`.
- Usar variáveis e métodos estáticos.
- Importar membros estáticos de uma classe.
- Usar o tipo `enum` para criar conjuntos de constantes com identificadores exclusivos.
- Declarar constantes `enum` com parâmetros.
- Organizar classes em pacotes para promover a reutilização.



- |                                                                                      |                                                       |
|--------------------------------------------------------------------------------------|-------------------------------------------------------|
| <b>F.1</b> Introdução                                                                | <b>F.7</b> Composição                                 |
| <b>F.2</b> Estudo de caso da classe <code>Time</code>                                | <b>F.8</b> Enumerações                                |
| <b>F.3</b> Controle de acesso a membros                                              | <b>F.9</b> Coleta de lixo                             |
| <b>F.4</b> Referência aos membros do objeto atual com <code>this</code>              | <b>F.10</b> Membros de classe estáticos               |
| <b>F.5</b> Estudo de caso da classe <code>Time</code> : construtores sobrecarregados | <b>F.11</b> Variáveis de instância <code>final</code> |
| <b>F.6</b> Construtores padrão e sem argumentos                                      | <b>F.12</b> Pacotes                                   |
|                                                                                      | <b>F.13</b> Acesso ao pacote                          |
|                                                                                      | <b>F.14</b> Para finalizar                            |

*Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios*

## F.1 Introdução

Agora examinamos com mais profundidade a construção de classes, o controle de acesso aos membros de uma classe e a criação de construtores. Discutimos a composição – uma capacidade que permite a uma classe ter referências para objetos de outras classes como membros. Lembre-se de que a seção D.10 apresentou o tipo `enum` básico para declarar um conjunto de constantes. Neste apêndice, discutimos a relação entre os tipos `enum` e as classes, demonstrando que, assim como uma classe, um objeto `enum` pode ser declarado em seu próprio arquivo com construtores, métodos e campos. O apêndice também discute com detalhes os membros de classe estáticos e as variáveis de instância `final`. Por último, explicamos como organizar classes em pacotes para ajudar a gerenciar aplicativos grandes e promover a reutilização; então, mostramos uma relação especial entre as classes do mesmo pacote.

## F.2 Estudo de caso da classe `Time`

Nosso primeiro exemplo consiste em duas classes – `Time1` (Fig. F.1) e `Time1Test` (Fig. F.2). A classe `Time1` representa a hora do dia. `Time1Test` é uma classe de aplicativo na qual o método `main` cria um objeto da classe `Time1` e chama seus métodos. Essas classes devem ser declaradas em arquivos *separados*, pois ambas são `public`. A saída desse programa aparece na Fig. F.2.

### **Declaração da classe `Time1`**

As variáveis de instância `private int hour, minute` e `second` da classe `Time1` (Fig. F.1, linhas 6 a 8) representam a hora no formato universal (formato 24 horas, no qual as horas estão no intervalo de 0 a 23). A classe `Time1` contém os métodos `public setTime` (linhas 12 a 25), `toUniversalString` (linhas 28 a 31) e `toString` (linhas 34 a 39). Esses métodos também são denominados **serviços public** ou **interface public** que a classe fornece para seus clientes.

### **Construtor padrão**

Neste exemplo, a classe `Time1` não declara um construtor; portanto, ela tem um construtor padrão, fornecido pelo compilador. Cada variável de instância recebe implicitamente o valor padrão 0 para um `int`. As variáveis de instância também podem ser inicializadas ao serem declaradas no corpo da classe, usando-se a mesma sintaxe de inicialização de uma variável local.

```

1 // Fig. F.1: Time1.java
2 // A declaração da classe Time1 mantém a hora no formato 24 horas.
3
4 public class Time1
5 {
6 private int hour; // 0 - 23
7 private int minute; // 0 - 59
8 private int second; // 0 - 59
9
10 // configura um novo valor de hora usando a hora universal;
11 // lança uma exceção se hour, minute ou second for inválido
12 public void setTime(int h, int m, int s)
13 {
14 // valida hour, minute e second
15 if ((h >= 0 && h < 24) && (m >= 0 && m < 60) &&
16 (s >= 0 && s < 60))
17 {
18 hour = h;
19 minute = m;
20 second = s;
21 } // fim de if
22 else
23 throw new IllegalArgumentException(
24 "hour, minute and/or second was out of range");
25 } // fim do método setTime
26
27 // converte em String no formato de hora universal (HH:MM:SS)
28 public String toUniversalString()
29 {
30 return String.format("%02d:%02d:%02d", hour, minute, second);
31 } // fim do método toUniversalString
32
33 // converte em String no formato de hora padrão (H:MM:SS AM ou PM)
34 public String toString()
35 {
36 return String.format("%d:%02d:%02d %s",
37 ((hour == 0 || hour == 12) ? 12 : hour % 12),
38 minute, second, (hour < 12 ? "AM" : "PM"));
39 } // fim do método toString
40 } // fim da classe Time1

```

**Figura F.1** A declaração da classe Time1 mantém a hora no formato 24 horas.

### Método setTime e lançamento de exceções

setTime (linhas 12 a 25) é um método public que declara três parâmetros int e os utiliza para configurar a hora. As linhas 15 e 16 testam cada argumento para determinar se o valor está no intervalo correto e, se estiver, as linhas 18 a 20 atribuem os valores às variáveis de instância hour, minute e second. O valor de hour deve ser maior ou igual a 0 e menor que 24, pois o formato de hora universal representa as horas como inteiros de 0 a 23 (por exemplo, 1 PM é a hora 13 e 11 PM é a hora 23; a meia-noite é a hora 0 e o meio-dia é a hora 12). Do mesmo modo, os valores de minute e second devem ser maiores ou iguais a 0 e menores que 60. Para valores fora desses intervalos, setTime **lança uma exceção** do tipo `IllegalArgumentException` (linhas 23 e 24), a qual notifica o código cliente de que um argumento inválido foi passado para o método. Conforme você aprendeu no Apêndice E, é possível usar try...catch para capturar exceções e tentar se recuperar delas, o que faremos na Fig. F.2. A **instrução throw** (linha 23) cria um novo



objeto de tipo `IllegalArgumentException`. Os parênteses após o nome da classe indicam uma chamada para o construtor de `IllegalArgumentException`. Neste caso, chamamos o construtor, que nos permite especificar uma mensagem de erro personalizada. Após o objeto de exceção ser criado, a instrução `throw` termina o método `setTime` imediatamente e a exceção é retornada para o código que tentou configurar a hora.

### **Método `toUniversalString`**

O método `toUniversalString` (linhas 28 a 31) não recebe argumentos e retorna uma `String` no formato de hora universal, consistindo em dois dígitos para hora, dois para minuto e dois para segundo. Por exemplo, se a hora fosse 1:30:07 PM, o método retornaria 13:30:07. A linha 30 utiliza o método estático `format` da classe `String` para retornar uma `String` contendo os valores de hora (`hour`), minuto (`minute`) e segundo (`second`) formatados, cada um com dois dígitos e possivelmente um 0 à esquerda (especificado com o flag 0). O método `format` é semelhante ao método `System.out.printf`, exceto que `format` retorna uma `String` formatada, em vez de exibi-la em uma janela de comando. A `String` formatada é retornada pelo método `toUniversalString`.

### **Método `toString`**

O método `toString` (linhas 34 a 39) não recebe argumentos e retorna uma `String` no formato de hora padrão, consistindo nos valores de hora, minuto e segundo separados por dois pontos e seguidos por AM ou PM (por exemplo, 1:27:06 PM). Assim como o método `toUniversalString`, `toString` utiliza o método estático `format` de `String` para formatar os minutos e segundos como valores de dois dígitos, com zeros à esquerda, se necessário. A linha 37 usa um operador condicional (`?:`) para determinar o valor da hora na `String` – se a hora é 0 ou 12 (AM ou PM), ela aparece como 12; caso contrário, aparece como um valor de 1 a 11. O operador condicional na linha 38 determina se AM ou PM será retornado como parte da `String`.

Lembre, da seção D.4, que em Java todos os objetos têm um método `toString` que retorna uma representação `String` do objeto. Optamos por retornar uma `String` contendo a hora no formato padrão. O método `toString` é chamado implicitamente quando um objeto `Time1` aparece no código onde uma `String` é necessária, como o valor a ser gerado com um especificador de formato `%s` em uma chamada para `System.out.printf`.

### **Usando a classe `Time1`**

Conforme você aprendeu no Apêndice B, cada classe declarada representa um novo *tipo* em Java. Portanto, após declararmos a classe `Time1`, podemos usá-la como um tipo em declarações como

```
Time1 sunset; // sunset pode armazenar uma referência para um objeto Time1
```

A classe de aplicativo `Time1Test` (Fig. F.2) utiliza a classe `Time1`. A linha 9 declara e cria um objeto `Time1` e o atribui à variável local `time`. O operador `new` chama implicitamente o construtor padrão da classe `Time1`, pois `Time1` não declara construtor. As linhas 12 a 16 geram a hora primeiramente no formato universal (chamando o método `toUniversalString` de `time` na linha 13) e, depois, no formato padrão (chamando explicitamente o método `toString` de `time` na linha 15) para confirmar que o objeto `Time1` foi inicializado corretamente. Em seguida, a linha 19 chama o método `setTime` do objeto `time` para alterar a hora. Então, as linhas 20 a 24 geram a hora novamente nos dois formatos para confirmar que ela foi configurada corretamente.

```

1 // Fig. F.2: Time1Test.java
2 // Objeto Time1 usado em um aplicativo.
3
4 public class Time1Test
5 {
6 public static void main(String[] args)
7 {
8 // cria e inicializa um objeto Time1
9 Time1 time = new Time1(); // invokes Time1 constructor
10
11 // gera as representações de string da hora
12 System.out.print("The initial universal time is: ");
13 System.out.println(time.toUniversalString());
14 System.out.print("The initial standard time is: ");
15 System.out.println(time.toString());
16 System.out.println(); // gera uma linha em branco na saída
17
18 // muda a hora e gera a hora atualizada na saída
19 time.setTime(13, 27, 6);
20 System.out.print("Universal time after setTime is: ");
21 System.out.println(time.toUniversalString());
22 System.out.print("Standard time after setTime is: ");
23 System.out.println(time.toString());
24 System.out.println(); // gera uma linha em branco na saída
25
26 // tenta configurar a hora com valores inválidos
27 try
28 {
29 time.setTime(99, 99, 99); // all values out of range
30 } // fim do try
31 catch (IllegalArgumentException e)
32 {
33 System.out.printf("Exception: %s\n\n", e.getMessage());
34 } // fim do catch
35
36 // exibe a hora depois de tentar configurar valores inválidos
37 System.out.println("After attempting invalid settings:");
38 System.out.print("Universal time: ");
39 System.out.println(time.toUniversalString());
40 System.out.print("Standard time: ");
41 System.out.println(time.toString());
42 } // fim de main
43 } // fim da classe Time1Test

```

```

The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM

Universal time after setTime is: 13:27:06
Standard time after setTime is: 1:27:06 PM

Exception: hour, minute and/or second was out of range

After attempting invalid settings:
Universal time: 13:27:06
Standard time: 1:27:06 PM

```

**Figura F.2** Objeto Time1 usado em um aplicativo.

### ***Chamando o método setTime de Time1 com valores inválidos***

Para ilustrar o fato de que o método setTime valida seus argumentos, a linha 29 o chama com argumentos 99 para hour, minute e second. Essa instrução é colocada em um bloco try (linhas 27 a 30) para o caso de setTime lançar uma exceção IllegalArgumentException, o que realmente fará, pois todos os argumentos são inválidos. Quando isso ocorre, a exceção é

capturada nas linhas 31 a 34, e a linha 33 exibe a mensagem de erro da exceção chamando seu método `getMessage`. As linhas 37 a 41 geram a hora novamente nos dois formatos para confirmar que `setTime` não mudou a hora quando foram fornecidos argumentos inválidos.

### Observações sobre a declaração da classe `Time1`

Considere várias questões de projeto com relação à classe `Time1`. As variáveis de instância `hour`, `minute` e `second` são declaradas como `private`. A representação de dados real utilizada dentro da classe não é uma preocupação para os clientes da classe. Por exemplo, seria perfeitamente razoável `Time1` representar a hora internamente como o número de segundos decorridos desde a meia-noite ou o número de minutos e segundos desde a meia-noite. Os clientes poderiam usar os mesmos métodos `public` e obter os mesmos resultados sem saber disso.

## F.3 Controle de acesso a membros

Os modificadores de acesso `public` e `private` controlam o acesso às variáveis e aos métodos de uma classe. No Apêndice G, apresentaremos o modificador de acesso `protected`. Como você sabe, a principal finalidade dos métodos `public` é apresentar para os clientes da classe uma visão dos serviços por ela fornecidos (a interface `public` da classe). Os clientes não precisam se preocupar com como a classe realiza suas tarefas. Por isso, as variáveis e os métodos `private` da classe (isto é, os detalhes de sua implementação) *não* são acessíveis para seus clientes.

A Figura F.3 demonstra que os membros da classe `private` não são acessíveis fora da classe. As linhas 9 a 11 tentam acessar diretamente as variáveis de instância `private` `hour`, `minute` e `second` do objeto `time` de `Time1`. Quando esse programa é compilado, o compilador gera mensagens de erro dizendo que esses membros `private` não são acessíveis. Esse programa presume que é usada a classe `Time1` da Fig. F.1.

```

1 // Fig. F.3: MemberAccessTest.java
2 // Os membros private da classe Time1 não são acessíveis.
3 public class MemberAccessTest
4 {
5 public static void main(String[] args)
6 {
7 Time1 time = new Time1(); // cria e inicializa objeto Time1
8
9 time.hour = 7; // erro: hour tem acesso private em Time1
10 time.minute = 15; // erro: minute tem acesso private em Time1
11 time.second = 30; // erro: second tem acesso private em Time1
12 } // fim de main
13 } // fim da classe MemberAccessTest

```

```

MemberAccessTest.java:9: hour has private access in Time1
 time.hour = 7; // error: hour has private access in Time1
 ^
MemberAccessTest.java:10: minute has private access in Time1
 time.minute = 15; // error: minute has private access in Time1
 ^
MemberAccessTest.java:11: second has private access in Time1
 time.second = 30; // error: second has private access in Time1
 ^
3 errors

```

**Figura F.3** Os membros `private` da classe `Time1` não são acessíveis.

## F.4 Referência aos membros do objeto atual com `this`

Todo objeto pode acessar uma referência para si mesmo com a palavra-chave `this` (às vezes chamada de **referência `this`**). Quando um método não estático é chamado por um objeto em particular, o corpo do método usa a palavra-chave `this` implicitamente para se referir às variáveis de instância e a outros métodos do objeto. Isso permite que o código da classe saiba qual objeto deve ser manipulado. Conforme você vai ver na Fig. F.4, também é possível usar a palavra-chave `this` explicitamente no corpo de um método não estático. A seção F.5 mostra outro uso interessante da palavra-chave `this`. A seção F.10 explica por que ela não pode ser usada em um método estático.

Agora, demonstramos o uso implícito e explícito da referência `this` (Fig. F.4). Esse exemplo é o primeiro no qual declaramos *duas* classes em um único arquivo – a classe `ThisTest` é declarada nas linhas 4 a 11 e a classe `SimpleTime` nas linhas 14 a 47. Fazemos isso para demonstrar que, quando você compila um arquivo `.java` contendo mais de uma classe, o compilador produz um arquivo de classe separado com a extensão `.class` para cada classe compilada. Neste caso, são produzidos dois arquivos separados – `SimpleTime.class` e `ThisTest.class`. Quando um único arquivo de código-fonte (`.java`) contém várias declarações de classe, o compilador coloca os dois arquivos dessas classes no mesmo diretório. Observe também, na Fig. F.4, que somente a classe `ThisTest` é declarada como `public`. Um arquivo de código-fonte pode conter somente uma classe `public` – caso contrário, ocorrerá um erro de compilação. Classes não `public` só podem ser usadas por outras classes do mesmo pacote. Assim, neste exemplo a classe `SimpleTime` só pode ser usada pela classe `ThisTest`.

```

1 // Fig. F.4: ThisTest.java
2 // this usada implícita e explicitamente para se referir aos membros de um objeto.
3
4 public class ThisTest
5 {
6 public static void main(String[] args)
7 {
8 SimpleTime time = new SimpleTime(15, 30, 19);
9 System.out.println(time.buildString());
10 } // fim de main
11 } // fim da classe ThisTest
12
13 // a classe SimpleTime demonstra a referência "this"
14 class SimpleTime
15 {
16 private int hour; // 0-23
17 private int minute; // 0-59
18 private int second; // 0-59
19
20 // se o construtor usa nomes de parâmetro idênticos aos
21 // nomes de variável de instância, a referência "this" é
22 // obrigatória, para diferenciá-los
23 public SimpleTime(int hour, int minute, int second)
24 {
25 this.hour = hour; // configura hour do objeto "this"
26 this.minute = minute; // configura minute do objeto "this"
27 this.second = second; // configura second do objeto "this"
28 } // fim do construtor de SimpleTime

```

**Figura F.4** `this` usada implícita e explicitamente para se referir aos membros de um objeto. (continua)

```

29
30 // usa "this" explícito e implícito para chamar toUniversalString
31 public String buildString()
32 {
33 return String.format("%24s: %s\n%24s: %s",
34 "this.toUniversalString()", this.toUniversalString(),
35 "toUniversalString()", toUniversalString());
36 } // fim do método buildString
37
38 // converte em String no formato de hora universal (HH:MM:SS)
39 public String toUniversalString()
40 {
41 // "this" não é exigida aqui para acessar variáveis de instância,
42 // pois o método não tem variáveis locais com os mesmos
43 // nomes das variáveis de instância
44 return String.format("%02d:%02d:%02d",
45 this.hour, this.minute, this.second);
46 } // fim do método toUniversalString
47 } // fim da classe SimpleTime

```

```

this.toUniversalString(): 15:30:19
toUniversalString(): 15:30:19

```

**Figura F.4** this usada implícita e explicitamente para se referir aos membros de um objeto.

A classe `SimpleTime` (linhas 14 a 47) declara três variáveis de instância `private` – `hour`, `minute` e `second` (linhas 16 a 18). O construtor da classe (linhas 23 a 28) recebe três argumentos `int` para inicializar o objeto `SimpleTime`. No construtor, usamos nomes de parâmetro (linha 23) idênticos aos nomes de variável de instância da classe (linhas 16 a 18). Não recomendamos essa prática, mas fizemos isso aqui para sombrear (ocultar) as variáveis de instância correspondentes, a fim de podermos ilustrar um caso em que o uso *explícito* da referência `this` é obrigatório. Se um método contém uma variável local com o *mesmo* nome de um campo, esse método vai se referir à variável local e não ao campo. Nesse caso, a variável local sombreia o campo no escopo do método. Contudo, o método pode usar a referência `this` para se referir ao campo sombreado explicitamente, como mostrado nos lados esquerdo das atribuições das linhas 25 a 27 para as variáveis de instância sombreadas de `SimpleTime`.

O método `buildString` (linhas 31 a 36) retorna uma `String` criada por uma instrução que utiliza a referência `this` explícita e implicitamente. A linha 34 a utiliza explicitamente para chamar o método `toUniversalString`. A linha 35 a utiliza implicitamente para chamar o mesmo método. As duas linhas executam a mesma tarefa. Normalmente, você não vai usar `this` explicitamente para fazer referência a outros métodos dentro do objeto atual. Além disso, a linha 45 no método `toUniversalString` utiliza a referência `this` explicitamente para acessar cada variável de instância. Isso *não* é necessário aqui, pois o método *não* tem uma variável local que sombreie as variáveis de instância da classe.



#### Erro de programação comum F.1

Frequentemente ocorre um erro de lógica quando um método contém um parâmetro ou uma variável local que tem o mesmo nome de um campo da classe. Nesse caso, use a referência `this` se quiser acessar o campo da classe – caso contrário, senão referenciado o parâmetro do método ou a variável local.



#### Dica de prevenção de erro F.1

*Evite nomes de parâmetro de método ou de variável local que entrem em conflito com nomes de campo. Isso ajuda a evitar erros sutis, difíceis de localizar.*



#### Dica de desempenho F.1

*A linguagem Java conserva espaço de armazenamento mantendo apenas uma cópia de cada método por classe – esse método é chamado por todo objeto da classe. Por outro lado, cada objeto tem sua própria cópia das variáveis de instância da classe (isto é, campos não estáticos). Cada método da classe usa `this` implicitamente para determinar o objeto específico da classe a ser manipulado.*

A classe de aplicativo `ThisTest` (linhas 4 a 11) demonstra a classe `SimpleTime`. A linha 8 cria uma instância da classe `SimpleTime` e chama seu construtor. A linha 9 chama o método `buildString` do objeto e, então, exibe os resultados.

## F.5 Estudo de caso da classe `Time`: construtores sobrecarregados

Como você sabe, é possível declarar seus próprios construtores para especificar como os objetos de uma classe devem ser inicializados. A seguir, demonstramos uma classe com vários **construtores sobrecarregados** que permitem inicializar objetos dessa classe de diferentes maneiras. Para sobrecarregar construtores, basta fornecer várias declarações de construtor com assinaturas diferentes.

### Classe `Time2` com construtores sobrecarregados

O construtor padrão da classe `Time1` (Fig. F.1) inicializou `hour`, `minute` e `second` com seus valores padrão 0 (que é meia-noite na hora universal). O construtor padrão não permite que os clientes da classe inicializem a hora com valores específicos diferentes de zero. A classe `Time2` (Fig. F.5) contém cinco construtores sobrecarregados que fornecem maneiras convenientes de inicializar objetos da nova classe `Time2`. Cada construtor inicializa o objeto de modo a começar em um estado coerente. Nesse programa, quatro dos construtores chamam um quinto, o qual, por sua vez, chama o método `setTime` para garantir que o valor fornecido para `hour` esteja no intervalo de 0 a 23 e que os valores de `minute` e `second` estejam no intervalo de 0 a 59. O compilador chama o construtor apropriado fazendo a correspondência do número, dos tipos e da ordem dos tipos dos argumentos especificados na chamada do construtor, com o número, os tipos e a ordem dos tipos dos parâmetros especificados na declaração de cada construtor. A classe `Time2` fornece também métodos `set` e `get` para cada variável de instância.

```

1 // Fig. F.5: Time2.java
2 // Classe Time2 com construtores sobrecarregados.
3
4 public class Time2
5 {
6 private int hour; // 0 - 23
7 private int minute; // 0 - 59
8 private int second; // 0 - 59
9

```

**Figura F.5** Classe `Time2` com construtores sobrecarregados. (continua)

```

10 // Construtor sem argumento de Time2:
11 // inicializa cada variável de instância com zero
12 public Time2()
13 {
14 this(0, 0, 0); // chama o construtor de Time2 com três argumentos
15 } // fim do construtor sem argumentos de Time2
16
17 // construtor de Time2: hour fornecida, minute e second tendo 0 como padrão
18 public Time2(int h)
19 {
20 this(h, 0, 0); // chama o construtor de Time2 com três argumentos
21 } // fim do construtor com um argumento de Time2
22
23 // construtor de Time2: hour e minute fornecidas, second tendo 0 como padrão
24 public Time2(int h, int m)
25 {
26 this(h, m, 0); // chama o construtor de Time2 com três argumentos
27 } // fim do construtor com dois argumentos de Time2
28
29 // construtor de Time2: hour, minute e second fornecidas
30 public Time2(int h, int m, int s)
31 {
32 setTime(h, m, s); // chama setTime para validar a hora
33 } // fim do construtor com três argumentos de Time2
34
35 // construtor de Time2: outro objeto Time2 fornecido
36 public Time2(Time2 time)
37 {
38 // chama o construtor com três argumentos de Time2
39 this(time.getHour(), time.getMinute(), time.getSecond());
40 } // fim do construtor de Time2 com um argumento de objeto Time2
41
42 // Métodos set
43 // configura um novo valor de hora usando a hora universal;
44 // valida os dados
45 public void setTime(int h, int m, int s)
46 {
47 setHour(h); // configura a hora
48 setMinute(m); // configura o minuto
49 setSecond(s); // configura o segundo
50 } // fim do método setTime
51
52 // valida e configura a hora
53 public void setHour(int h)
54 {
55 if (h >= 0 && h < 24)
56 hour = h;
57 else
58 throw new IllegalArgumentException("hour must be 0-23");
59 } // fim do método setHour
60
61 // valida e configura o minuto
62 public void setMinute(int m)
63 {
64 if (m >= 0 && m < 60)
65 minute = m;
66 else
67 throw new IllegalArgumentException("minute must be 0-59");
68 } // fim do método setMinute
69
70 // valida e configura o segundo
71 public void setSecond(int s)

```

**Figura F.5** Classe Time2 com construtores sobrecarregados.

```

72 {
73 if (s >= 0 && s < 60)
74 second = ((s >= 0 && s < 60) ? s : 0);
75 else
76 throw new IllegalArgumentException("second must be 0-59");
77 } // fim do método setSecond
78
79 // Métodos get
80 // obtém o valor da hora
81 public int getHour()
82 {
83 return hour;
84 } // fim do método getHour
85
86 // obtém o valor do minuto
87 public int getMinute()
88 {
89 return minute;
90 } // fim do método getMinute
91
92 // obtém o valor do segundo
93 public int getSecond()
94 {
95 return second;
96 } // fim do método getSecond
97
98 // converte em String no formato de hora universal (HH:MM:SS)
99 public String toUniversalString()
100 {
101 return String.format(
102 "%02d:%02d:%02d", getHour(), getMinute(), getSecond());
103 } // fim do método toUniversalString
104
105 // converte em String no formato de hora padrão (H:MM:SS AM ou PM)
106 public String toString()
107 {
108 return String.format("%d:%02d:%02d %s",
109 (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ,
110 getMinute(), getSecond(), (getHour() < 12 ? "AM" : "PM"));
111 } // fim do método toString
112 } // fim da classe Time2

```

**Figura F.5** Classe Time2 com construtores sobrecarregados.

### Construtores da classe Time2

As linhas 12 a 15 declaram o assim chamado **construtor sem argumentos** que é chamado, portanto, sem argumentos. Quando você declarar quaisquer construtores em uma classe, o compilador *não* vai fornecer um construtor padrão. Esse construtor sem argumentos garante que os clientes da classe Time2 possam criar objetos Time2 com valores padrão. Tal construtor simplesmente inicializa o objeto conforme especificado em seu corpo. Nele, apresentamos um uso da referência this que só é permitida como a *primeira* instrução no corpo de um construtor. A linha 14 utiliza this na sintaxe da chamada de método para ativar o construtor de Time2 que recebe três parâmetros (linhas 30 a 33) com valores 0 para hour, minute e second. Usar a referência this como mostrado aqui é uma maneira popular de reutilizar o código de inicialização fornecido por outro dos construtores da classe, em vez de definir código semelhante no corpo do construtor sem argumentos. Usamos essa sintaxe em quatro dos cinco construtores de Time2 para tornar a classe mais fácil de manter e modificar. Se precisarmos mudar o modo como os objetos da classe Time2 são inicializados, somente o construtor chamado pelos outros construtores da classe precisará ser



modificado. Na verdade, mesmo esse construtor talvez não precise de modificação neste exemplo. Ele simplesmente chama o método `setTime` para fazer a inicialização; portanto, é possível que as alterações que a classe possa exigir estejam localizadas nos métodos `set`.



### Erro de programação comum F.2

*Ocorre um erro de compilação quando `this` é usado no corpo de um construtor para chamar outro construtor da mesma classe se essa chamada não for a primeira instrução no construtor. Também ocorre um erro de compilação quando um método tenta chamar um construtor diretamente, via `this`.*



### Erro de programação comum F.3

*Um construtor pode chamar métodos da classe. Saiba que as variáveis de instância podem ainda não estar inicializadas, pois o construtor está no processo de inicialização do objeto. Usar variáveis de instância antes que elas sejam inicializadas corretamente é um erro de lógica.*

As linhas 18 a 21 declaram um construtor de `Time2` com um único parâmetro `int` representando `hour`, o qual é passado para o construtor nas linhas 30 a 33 com 0 para `minute` e `second`. As linhas 24 a 27 declaram um construtor de `Time2` que recebe dois parâmetros `int` representando `hour` e `minute`, os quais são passados para o construtor nas linhas 30 a 33 com 0 para `second`. Assim como o construtor sem argumentos, cada um desses construtores chama o construtor das linhas 30 a 33 para minimizar a duplicação de código. As linhas 30 a 33 declaram o construtor de `Time2` que recebe três parâmetros `int` representando `hour`, `minute` e `second`. Esse construtor chama `setTime` para inicializar as variáveis de instância.

As linhas 36 a 40 declaram um construtor de `Time2` que recebe uma referência para outro objeto `Time2`. Neste caso, os valores do argumento de `Time2` são passados para o construtor de três argumentos nas linhas 30 a 33, para inicializar `hour`, `minute` e `second`. A linha 39 poderia ter acessado diretamente os valores de `hour`, `minute` e `second`, do argumento `time` do construtor, com as expressões `time.hour`, `time.minute` e `time.second` — mesmo `hour`, `minute` e `second` sendo declarados como variáveis `private` da classe `Time2`. Isso se dá devido à relação especial entre objetos da mesma classe. Em breve, vamos ver por que é preferível usar os métodos `get`.



### Observação sobre engenharia de software F.1

*Quando um objeto de uma classe tem uma referência para outro objeto da mesma classe, o primeiro pode acessar todos os dados e métodos do segundo objeto (inclusive os que são `private`).*

### Método `setTime` da classe `Time2`

O método `setTime` (linhas 45 a 50) chama os métodos `setHour` (linhas 53 a 59), `setMinute` (linhas 62 a 68) e `setSecond` (linhas 71 a 77), os quais garantem que o valor fornecido para `hour` esteja no intervalo de 0 a 23 e que os valores de `minute` e `second` estejam no intervalo de 0 a 59. Se um valor está fora do intervalo, cada um desses métodos lança uma exceção `IllegalArgumentException` (linhas 58, 67 e 76) indicando qual valor estava fora do intervalo.

### Observações a respeito dos métodos `set` e `get` e construtores da classe `Time2`

Os métodos `set` e `get` de `Time2` são chamados por toda a classe. Em particular, o método `setTime` chama os métodos `setHour`, `setMinute` e `setSecond` nas linhas 47 a 49, e os méto-

dos `toUniversalString` e `toString` chamam os métodos `getHour`, `getMinute` e `getSecond` na linha 102 e nas linhas 109 e 110, respectivamente. Em cada caso, esses métodos poderiam ter acesso direto aos dados `private` da classe, sem chamar os métodos *set* e *get*. Contudo, pense na hipótese de mudar a representação da hora de três valores `int` (exigindo 12 bytes de memória) para um único valor `int` representando o número total de segundos decorridos desde a meia-noite (exigindo apenas 4 bytes de memória). Se fizéssemos essa alteração, somente os corpos dos métodos que acessam os dados `private` diretamente precisariam mudar – em particular, os métodos *set* e *get* individuais para `hour`, `minute` e `second`. Não haveria necessidade de modificar os corpos dos métodos `setTime`, `toUniversalString` ou `toString`, pois eles não acessam os dados diretamente. Projetar a classe dessa maneira reduz a probabilidade de erros de programação ao se alterar a implementação da classe.

Do mesmo modo, cada construtor de `Time2` poderia incluir uma cópia das instruções apropriadas dos métodos `setHour`, `setMinute` e `setSecond`. Isso pode ser um pouco mais eficiente, pois as chamadas extras para o construtor e para `setTime` são eliminadas. No entanto, *duplicar* instruções em vários métodos ou construtores torna mais difícil alterar a representação de dados interna da classe. Fazer que os construtores de `Time2` chamem o construtor com três argumentos (ou mesmo chamem `setTime` diretamente) faz que quaisquer alterações na implementação de `setTime` sejam feitas apenas uma vez. Além disso, o compilador pode otimizar programas, removendo chamadas para métodos simples e substituindo-as pelo código expandido de suas declarações – uma técnica conhecida como **codificação em linha**, a qual melhora o desempenho do programa.



#### Observação sobre engenharia de software F.2

*Ao implementar um método de uma classe, use os métodos *set* e *get* da classe para acessar os dados *private* da classe. Isso simplifica a manutenção do código e reduz a probabilidade de erros.*

### Usando os construtores sobrecarregados da classe `Time2`

A classe `Time2Test` (Fig. F.6) chama os construtores sobrecarregados de `Time2` (linhas 8 a 12 e 40). A linha 8 chama o construtor sem argumentos (Fig. F.5, linhas 12 a 15). As linhas 9 a 13 do programa demonstram a passagem de argumentos para os outros construtores de `Time2`. A linha 9 chama o construtor de um argumento que recebe um valor `int` nas linhas 18 a 21 da Fig. F.5. A linha 10 chama o construtor de dois argumentos nas linhas 24 a 27 da Fig. F.5. A linha 11 chama o construtor de três argumentos nas linhas 30 a 33 da Fig. F.5. A linha 12 chama o construtor de um argumento que recebe um objeto `Time2` nas linhas 36 a 40 da Fig. F.5. Em seguida, o aplicativo exibe as representações de `String` de cada objeto `Time2` para confirmar que foram inicializados corretamente. A linha 40 tenta inicializar `t6` criando um novo objeto `Time2` e passando três valores inválidos para o construtor. Quando o construtor tenta usar o valor de hora inválido para inicializar `hour` do objeto, ocorre uma exceção `IllegalArgumentException`. Capturamos essa exceção na linha 42 e exibimos sua mensagem de erro, a qual resulta na última linha da saída.

```

1 // Fig. F.6: Time2Test.java
2 // Construtores sobrecarregados usados para inicializar objetos Time2.
3
4 public class Time2Test
5 {
6 public static void main(String[] args)

```

**Figura F.6** Construtores sobrecarregados usados para inicializar objetos `Time2`. (continua)

```

7 {
8 Time2 t1 = new Time2(); // 00:00:00
9 Time2 t2 = new Time2(2); // 02:00:00
10 Time2 t3 = new Time2(21, 34); // 21:34:00
11 Time2 t4 = new Time2(12, 25, 42); // 12:25:42
12 Time2 t5 = new Time2(t4); // 12:25:42
13
14 System.out.println("Constructed with:");
15 System.out.println("t1: all arguments defaulted");
16 System.out.printf(" %s\n", t1.toUniversalString());
17 System.out.printf(" %s\n", t1.toString());
18
19 System.out.println(
20 "t2: hour specified; minute and second defaulted");
21 System.out.printf(" %s\n", t2.toUniversalString());
22 System.out.printf(" %s\n", t2.toString());
23
24 System.out.println(
25 "t3: hour and minute specified; second defaulted");
26 System.out.printf(" %s\n", t3.toUniversalString());
27 System.out.printf(" %s\n", t3.toString());
28
29 System.out.println("t4: hour, minute and second specified");
30 System.out.printf(" %s\n", t4.toUniversalString());
31 System.out.printf(" %s\n", t4.toString());
32
33 System.out.println("t5: Time2 object t4 specified");
34 System.out.printf(" %s\n", t5.toUniversalString());
35 System.out.printf(" %s\n", t5.toString());
36
37 // tenta inicializar t6 com valores inválidos
38 try
39 {
40 Time2 t6 = new Time2(27, 74, 99); // valores inválidos
41 } // fim de try
42 catch (IllegalArgumentException e)
43 {
44 System.out.printf("\nException while initializing t6: %s\n",
45 e.getMessage());
46 } // fim do catch
47 } // fim de main
48 } // fim da classe Time2Test

```

```

Constructed with:
t1: all arguments defaulted
 00:00:00
 12:00:00 AM
t2: hour specified; minute and second defaulted
 02:00:00
 2:00:00 AM
t3: hour and minute specified; second defaulted
 21:34:00
 9:34:00 PM
t4: hour, minute and second specified
 12:25:42
 12:25:42 PM
t5: Time2 object t4 specified
 12:25:42
 12:25:42 PM

Exception while initializing t6: hour must be 0-23

```

**Figura F.6** Construtores sobrecarregados usados para inicializar objetos Time2.

## F.6 Construtores padrão e sem argumentos

Toda classe deve ter pelo menos um construtor. Se você não fornecer um na declaração de uma classe, o compilador criará um construtor padrão que não recebe argumentos ao ser chamado. O construtor padrão inicializa as variáveis de instância com os valores iniciais especificados em suas declarações ou com seus valores padrão (zero para tipos numéricos primitivos, false para valores boolean e null para referências). Na seção G.4.1, você vai aprender que o construtor padrão também executa outra tarefa.

Se sua classe declara construtores, o compilador *não* criará um construtor padrão. Nesse caso, você deve declarar um construtor sem argumentos se a inicialização padrão for exigida. Assim como um construtor padrão, um construtor sem argumentos é chamado com parênteses vazios. O construtor sem argumentos de Time2 (linhas 12 a 15 da Fig. F.5) inicializa explicitamente um objeto Time2, passando 0 para cada parâmetro do construtor de três argumentos. Como 0 é o valor padrão para variáveis de instância int, o construtor sem argumentos deste exemplo poderia, na verdade, ser declarado com um corpo vazio. Nesse caso, cada variável de instância receberia seu valor padrão quando o construtor sem argumentos fosse chamado. Se omitíssemos o construtor sem argumentos, os clientes dessa classe não poderiam criar um objeto Time2 com a expressão new Time2().

## F.7 Composição

Uma classe pode ter referências para objetos de outras classes como membros. Isso se chama **composição** e às vezes é referido como **relação tem um**. Por exemplo, um objeto AlarmClock (despertador) precisa saber a hora atual e a hora em que deve soar seu alarme, de modo que é razoável incluir *duas* referências a objetos Time (tempo) em um objeto AlarmClock.

### Classe Date

Este exemplo de composição contém as classes Date (Fig. F.7), Employee (Fig. F.8) e EmployeeTest (Fig. F.9). A classe Date (Fig. F.7) declara as variáveis de instância month, day e year (linhas 6 a 8) para representar uma data. O construtor recebe três parâmetros int. A linha 17 chama o método utilitário checkMonth (linhas 26 a 32) para validar o mês (month) – se o valor estiver fora do intervalo, o método lança uma exceção. A linha 15 presume que o valor de year (ano) está correto e não o valida. A linha 19 chama o método utilitário checkDay (linhas 35 a 48) para validar o dia (day) com base no mês e no ano correntes. A linha 38 determina se o dia está correto com base no número de dias no mês em particular. Se o dia não está correto, as linhas 42 e 43 determinam se o mês é fevereiro, se o dia é 29 e se o ano é bissexto. Se o dia ainda é inválido, o método lança uma exceção. As linhas 21 e 22 do construtor geram a referência this como uma String na saída. Como this é uma referência para o objeto Date atual, o método toString do objeto (linhas 51 a 54) é chamado *implicitamente* para obter a representação de String do objeto.

```

1 // Fig. F.7: Date.java
2 // Declaração da classe Date.
3
4 public class Date
5 {
6 private int month; // 1-12
7 private int day; // 1-31 de acordo com o mês
8 private int year; // qualquer ano

```

**Figura F.7** Declaração da classe Date. (continua)

```

9
10 private static final int[] daysPerMonth = // dias em cada mês
11 { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
12
13 // construtor: chama checkMonth para confirmar o valor correto do mês;
14 // chama checkDay para confirmar o valor correto do dia
15 public Date(int theMonth, int theDay, int theYear)
16 {
17 month = checkMonth(theMonth); // valida o mês
18 year = theYear; // pode validar o ano
19 day = checkDay(theDay); // valida o dia
20
21 System.out.printf(
22 "Date object constructor for date %s\n", this);
23 } // fim do construtor de Date
24
25 // método utilitário para confirmar o valor correto do mês
26 private int checkMonth(int testMonth)
27 {
28 if (testMonth > 0 && testMonth <= 12) // valida o mês
29 return testMonth;
30 else // o mês é inválido
31 throw new IllegalArgumentException("month must be 1-12");
32 } // fim do método checkMonth
33
34 // método utilitário para confirmar o valor correto do dia com base no mês e no ano
35 private int checkDay(int testDay)
36 {
37 // verifica se o dia está no intervalo do mês
38 if (testDay > 0 && testDay <= daysPerMonth[month])
39 return testDay;
40
41 // verifica se é ano bissexto
42 if (month == 2 && testDay == 29 && (year % 400 == 0 ||
43 (year % 4 == 0 && year % 100 != 0)))
44 return testDay;
45
46 throw new IllegalArgumentException(
47 "day out-of-range for the specified month and year");
48 } // fim do método checkDay
49
50 // retorna uma String da forma mês/dia/ano
51 public String toString()
52 {
53 return String.format("%d/%d/%d", month, day, year);
54 } // fim do método toString
55 } // fim da classe Date

```

**Figura F.7** Declaração da classe Date.

### Classe Employee

A classe Employee (Fig. F.8) tem as variáveis de instância `firstName`, `lastName`, `birthDate` e `hireDate`. Os membros `firstName` e `lastName` (linhas 6 e 7) são referências para objetos `String`. Os membros `birthDate` e `hireDate` (linhas 8 e 9) são referências para objetos `Date`. Isso demonstra que uma classe pode ter, como variáveis de instância, referências para objetos de outras classes. O construtor de `Employee` (linhas 12 a 19) recebe quatro parâmetros – `first`, `last`, `dateOfBirth` e `dateOfHire`. Os objetos referenciados pelos parâmetros são atribuídos às variáveis de instância do objeto `Employee`. Quando o método `toString` da classe `Employee` é chamado, ele retorna uma `String` contendo o nome do funcionário e as representações de `String` dos dois objetos `Date`. Cada uma dessas `Strings` é obtida com uma chamada *implícita* ao método `toString` da classe `Date`.

```

1 // Fig. F.8: Employee.java
2 // Classe Employee com referências para outros objetos.
3
4 public class Employee
5 {
6 private String firstName;
7 private String lastName;
8 private Date birthDate;
9 private Date hireDate;
10
11 // construtor para inicializar o nome, a data de nascimento e a data de contratação
12 public Employee(String first, String last, Date dateOfBirth,
13 Date dateOfHire)
14 {
15 firstName = first;
16 lastName = last;
17 birthDate = dateOfBirth;
18 hireDate = dateOfHire;
19 } // fim do construtor de Employee
20
21 // converte Employee para formato de String
22 public String toString()
23 {
24 return String.format("%s, %s Hired: %s Birthday: %s",
25 lastName, firstName, hireDate, birthDate);
26 } // fim do método toString
27 } // fim da classe Employee

```

**Figura F.8** Classe Employee com referências para outros objetos.

### Classe EmployeeTest

A classe EmployeeTest (Fig. F.9) cria dois objetos Date (linhas 8 e 9) para representar a data de nascimento (birthday) e a data de contratação (hire date) de um funcionário, respectivamente. A linha 10 cria um objeto Employee e inicializa suas variáveis de instância passando para o construtor duas Strings (representando o nome e o sobrenome do funcionário) e dois objetos Date (representando a data de nascimento e a data de contratação). A linha 12 chama implicitamente o método toString de Employee para exibir os valores de suas variáveis de instância e demonstrar que o objeto foi inicializado corretamente.

```

1 // Fig. F.9: EmployeeTest.java
2 // Demonstração de composição.
3
4 public class EmployeeTest
5 {
6 public static void main(String[] args)
7 {
8 Date birth = new Date(7, 24, 1949);
9 Date hire = new Date(3, 12, 1988);
10 Employee employee = new Employee("Bob", "Blue", birth, hire);
11
12 System.out.println(employee);
13 } // fim de main
14 } // fim da classe EmployeeTest

```

```

Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Blue, Bob Hired: 3/12/1988 Birthday: 7/24/1949

```

**Figura F.9** Demonstração de composição.

## F.8 Enumerações

Na Fig. D.5, apresentamos o tipo `enum` básico, o qual define um conjunto de constantes representadas como identificadores exclusivos. Naquele programa, as constantes `enum` representavam o status de um jogo. Nesta seção, discutimos a relação entre os tipos `enum` e as classes. Como as classes, todos os tipos `enum` são tipos de referência. Um tipo `enum` é declarado com uma **declaração `enum`**, que é uma lista de constantes `enum` separadas por vírgulas – opcionalmente, a declaração pode incluir outros componentes das classes tradicionais, como construtores, campos e métodos. Cada declaração `enum` declara uma classe `enum` com as seguintes restrições:

1. As constantes `enum` são implicitamente `final`, pois declaram constantes que não devem ser modificadas.
2. As constantes `enum` são implicitamente estáticas.
3. Qualquer tentativa de criar um objeto de tipo `enum` com o operador `new` resulta em um erro de compilação.

As constantes `enum` podem ser usadas em qualquer lugar onde constantes possam ser usadas, como nos rótulos `case` de instruções `switch` e para controlar instruções `for` melhoradas.

A Figura F.10 ilustra como declarar variáveis de instância, um construtor e métodos em um tipo `enum`. A declaração `enum` (linhas 5 a 37) contém duas partes – as constantes `enum` e os outros membros do tipo `enum`. A primeira parte (linhas 8 a 13) declara seis constantes `enum`. Opcionalmente, cada uma é seguida por argumentos, os quais são passados para o **construtor `enum`** (linhas 20 a 24). Assim como os construtores que vimos nas classes, um construtor `enum` pode especificar qualquer número de parâmetros e pode ser sobrecarregado. Neste exemplo, o construtor `enum` exige dois parâmetros `String`. Para inicializar cada constante `enum` corretamente, colocamos depois delas parênteses contendo dois argumentos `String`, os quais são passados para o construtor do `enum`. A segunda parte (linhas 16 a 36) declara os outros membros do tipo `enum` – duas variáveis de instância (linhas 16 e 17), um construtor (linhas 20 a 24) e dois métodos (linhas 27 a 30 e 33 a 36).

```

1 // Fig. F.10: Book.java
2 // Declarando um tipo enum com construtor, campos de instância
3 // explícitos e métodos de acesso para esses campos
4
5 public enum Book
6 {
7 // declara constantes do tipo enum
8 JHTP("Java How to Program", "2012"),
9 CHTP("C How to Program", "2007"),
10 IW3HTP("Internet & World Wide Web How to Program", "2008"),
11 CPPHTP("C++ How to Program", "2012"),
12 VBHTP("Visual Basic 2010 How to Program", "2011"),
13 CSHARPHTP("Visual C# 2010 How to Program", "2011");
14
15 // campos de instância
16 private final String title; // título do livro
17 private final String copyrightYear; // ano dos direitos autorais
18
19 // construtor enum

```

**Figura F.10** Declarando um tipo `enum` com construtor, campos de instância explícitos e métodos de acesso para esses campos.

```

20 Book(String bookTitle, String year)
21 {
22 title = bookTitle;
23 copyrightYear = year;
24 } // fim do construtor enum Book
25
26 // método de acesso para o campo title
27 public String getTitle()
28 {
29 return title;
30 } // fim do método getTitle
31
32 // método de acesso para o campo copyrightYear
33 public String getCopyrightYear()
34 {
35 return copyrightYear;
36 } // fim do método getCopyrightYear
37 } // fim do enum Book

```

**Figura F.10** Declarando um tipo enum com construtor, campos de instância explícitos e métodos de acesso para esses campos.

As linhas 16 e 17 declaram as variáveis de instância `title` e `copyrightYear`. Cada constante enum em `Book` é na verdade um objeto de tipo `Book` que tem sua própria cópia das variáveis de instância `title` e `copyrightYear`. O construtor (linhas 20 a 24) recebe dois parâmetros `String`, um que especifica o título do livro (book's title) e outro que especifica o ano de seus direitos autorais (copyright). As linhas 22 e 23 atribuem esses parâmetros às variáveis de instância. As linhas 27 a 36 declaram dois métodos, os quais retornam o título do livro e o ano dos direitos autorais, respectivamente.

A Figura F.11 testa o tipo enum `Book` e ilustra como iterar por um intervalo de constantes enum. Para cada enum, o compilador gera o método estático `values` (chamado na linha 12) que retorna um array das constantes de enum na ordem em que foram declaradas. As linhas 12 a 14 usam a instrução `for` melhorada para exibir todas as constantes declaradas no enum `Book`. A linha 14 chama os métodos `getTitle` e `getCopyrightYear` do enum `Book` para obter o título e o ano dos direitos autorais associados à constante. Quando uma constante enum é convertida em uma `String` (por exemplo, `book` na linha 13), o identificador da constante é usado como representação de `String` (por exemplo, `JHTP` para a primeira constante enum).

```

1 // Fig. F.11: EnumTest.java
2 // Testando o tipo enum Book.
3 import java.util.EnumSet;
4
5 public class EnumTest
6 {
7 public static void main(String[] args)
8 {
9 System.out.println("All books:\n");
10
11 // imprime todos os livros no enum Book
12 for (Book book : Book.values())
13 System.out.printf("%-10s%-45s\n", book,
14 book.getTitle(), book.getCopyrightYear());
15
16 System.out.println("\nDisplay a range of enum constants:\n");

```

**Figura F.11** Testando o tipo enum `Book`. (continua)



```

17
18 // imprime os quatro primeiros livros
19 for (Book book : EnumSet.range(Book.JHTP, Book.CPPHTP))
20 System.out.printf("%-10s%-45s%s\n", book,
21 book.getTitle(), book.getCopyrightYear());
22 } // fim de main
23 } // fim da classe EnumTest

```

All books:

|           |                                          |      |
|-----------|------------------------------------------|------|
| JHTP      | Java How to Program                      | 2012 |
| CHTP      | C How to Program                         | 2007 |
| IW3HTP    | Internet & World Wide Web How to Program | 2008 |
| CPPHTP    | C++ How to Program                       | 2012 |
| VBHTP     | Visual Basic 2010 How to Program         | 2011 |
| CSHARPHTP | Visual C# 2010 How to Program            | 2011 |

Display a range of enum constants:

|        |                                          |      |
|--------|------------------------------------------|------|
| JHTP   | Java How to Program                      | 2012 |
| CHTP   | C How to Program                         | 2007 |
| IW3HTP | Internet & World Wide Web How to Program | 2008 |
| CPPHTP | C++ How to Program                       | 2012 |

**Figura F.11** Testando o tipo enum Book.

As linhas 19 a 21 usam o método estático **range** da classe **EnumSet** (declarada no pacote `java.util`) para exibir um intervalo das constantes do enum **Book**. O método **range** recebe dois parâmetros – a primeira e a última constantes enum do intervalo – e retorna um **EnumSet** contendo todas as constantes entre essas duas constantes, inclusive. Por exemplo, a expressão `EnumSet.range( Book.JHTP, Book.CPPHTP )` retorna um **EnumSet** contendo **Book.JHTP**, **Book.CHTP**, **Book.IW3HTP** e **Book.CPPHTP**. A instrução **for** melhorada pode ser usada com um **EnumSet** exatamente como com um array; portanto, as linhas 12 a 14 a utilizam para exibir o título e o ano dos direitos autorais de cada livro no **EnumSet**. A classe **EnumSet** fornece vários outros métodos estáticos para criar conjuntos de constantes enum do mesmo tipo enum.



#### Erro de programação comum F.4

*Em uma declaração enum, é um erro de sintaxe declarar constantes enum depois dos construtores, campos e métodos do tipo enum.*

## F.9 Coleta de lixo

Todo objeto utiliza recursos do sistema, como a memória. Precisamos de uma maneira disciplinada de devolver recursos para o sistema quando não forem mais necessários; caso contrário, podem ocorrer “vazamentos de recurso” que os impediriam de ser reutilizados por seu programa ou possivelmente por outros programas. A JVM faz uma **coleta de lixo** automática para recuperar a memória ocupada por objetos que não são mais usados. Quando não existem mais referências para um objeto, o objeto está qualificado para ser coletado. Isso normalmente ocorre quando a JVM executa seu **coletor de lixo**. Assim, os vazamentos de memória, comuns em outras linguagens como C e C++ (porque a memória não é reclamada automaticamente nessas linguagens), são menos prováveis em Java – mas alguns ainda podem acontecer de maneiras sutis. Podem ocorrer outros tipos de vazamentos de recurso. Por exemplo, um aplicativo pode abrir um arquivo no disco

para modificar seu conteúdo. Se ele não fechar o arquivo, o aplicativo precisará terminar antes que qualquer outro aplicativo possa usá-lo.



#### Observação sobre engenharia de software F.3

*Uma classe que usa recursos de sistema, como arquivos no disco, deve fornecer um método que os programadores possam chamar para liberar os recursos quando não forem mais necessários em um programa. Muitas classes da API Java fornecem métodos `close` ou `dispose` com essa finalidade. Por exemplo, a classe `Scanner` tem um método `close`.*

## F.10 Membros de classe estáticos

Todo objeto tem sua própria cópia de todas as variáveis de instância da classe. Em certos casos, somente uma cópia de uma variável em particular deve ser *compartilhada* por todos os objetos de uma classe. Nesses casos, é usado um **campo estático** – chamado de **variável de classe**. Uma variável estática representa **informações em nível de classe** – todos os objetos da classe compartilham os *mesmos* dados. A declaração de uma variável estática começa com a palavra-chave `static`.

Vamos dar a motivação para os dados estáticos com um exemplo. Suponha que temos um videogame com marcianos e outras criaturas espaciais. Cada marciano (Marian) tende a ficar valente e disposto para atacar outras criaturas do espaço ao saber que pelo menos outros quatro marcianos (martians) estão presentes. Se menos de cinco marcianos estão presentes, cada um deles se torna covarde. Assim, cada marciano precisa saber qual é a contagem de marcianos (martianCount). Poderíamos dotar a classe Marian com uma variável de instância `martianCount`. Se fizermos isso, todo marciano terá *uma cópia separada* da variável de instância, e sempre que criarmos um novo marciano, precisaremos atualizar a variável de instância `martianCount` em cada objeto Marian. Isso desperdiça espaço com as cópias redundantes, desperdiça tempo na atualização das cópias separadas e é propenso a erros. Em vez disso, declaramos `martianCount` como `static`, tornando `martianCount` um dado em nível de classe. Todo objeto Marian pode ver a `martianCount` como se fosse uma variável de instância da classe Marian, mas é mantida somente uma cópia de `static martianCount`. Isso economiza espaço. Economizamos tempo fazendo o construtor de Marian incrementar `static martianCount` – existe apenas uma cópia; portanto, não precisamos incrementar cópias separadas de cada objeto Marian.



#### Observação sobre engenharia de software F.4

*Use uma variável estática quando todos os objetos de uma classe precisarem usar a mesma cópia da variável.*

As variáveis estáticas têm escopo de classe. Podemos acessar os membros `public static` de uma classe por meio de uma referência a qualquer objeto da classe ou qualificando o nome do membro com o nome da classe e um ponto (`.`), como em `Math.random()`. Os membros `private static` de uma classe podem ser acessados pelo código cliente somente por meio de métodos da classe. Na verdade, *existem membros de classe estáticos mesmo quando não existe objeto da classe* – eles estão disponíveis assim que a classe é carregada na memória, no momento da execução. Para acessar um membro `public static` quando não existe objeto da classe (e mesmo quando existem), prefixe o nome da classe e um ponto (`.`) ao membro estático, como em `Math.PI`. Para acessar um

membro `private static` quando não existe objeto da classe, forneça um método `public static` e chame-o qualificando seu nome com o nome da classe e um ponto.



#### Observação sobre engenharia de software F.5

*As variáveis e os métodos de classe estáticos existem e podem ser usados mesmo que nenhum objeto dessa classe tenha sido instanciado.*

Um método estático não pode acessar membros de classe não estáticos, pois um método estático pode ser chamado mesmo quando nenhum objeto da classe tiver sido instanciado. Pelo mesmo motivo, a referência `this` não pode ser usada em um método estático. Ela deve fazer referência a um objeto específico da classe e, quando um método estático é chamado, pode não haver objeto de sua classe na memória.



#### Erro de programação comum F.5

*Se um método estático chama um método de instância (não estático) na mesma classe usando somente o nome do método, ocorre um erro de compilação. Do mesmo modo, ocorre um erro de compilação se um método estático tenta acessar uma variável de instância na mesma classe usando somente o nome da variável.*



#### Erro de programação comum F.6

*Referir-se a `this` em um método estático é um erro de compilação.*

### Monitorando o número de objetos `Employee` criados

Nosso próximo programa declara duas classes – `Employee` (Fig. F.12) e `EmployeeTest` (Fig. F.13). A classe `Employee` declara a variável `private static count` (Fig. F.12, linha 9) e o método `public static getCount` (linhas 36 a 39). A variável `static count` é inicializada com zero na linha 9. Se uma variável `static` não é inicializada, o compilador atribui a ela um valor padrão – neste caso 0, o valor padrão para o tipo `int`. A variável `count` mantém a contagem do número de objetos da classe `Employee` criados até o momento.

```

1 // Fig. F.12: Employee.java
2 // Variável static usada para manter a contagem do número de
3 // objetos Employee na memória.
4
5 public class Employee
6 {
7 private String firstName;
8 private String lastName;
9 private static int count = 0; // número de objetos Employee criados
10
11 // inicializa Employee, soma 1 à variável static count e
12 // gera uma String indicando que o construtor foi chamado
13 public Employee(String first, String last)
14 {
15 firstName = first;
16 lastName = last;
17
18 ++count; // incrementa a variável static count de funcionários
19 System.out.printf("Employee constructor: %s %s; count = %d\n",

```

**Figura F.12** Variável `static` usada para manter a contagem do número de objetos `Employee` na memória.

```

20 firstName, lastName, count);
21 } // fim do construtor de Employee
22
23 // obtém o nome
24 public String getFirstName()
25 {
26 return firstName;
27 } // fim do método getFirstName
28
29 // obtém o sobrenome
30 public String getLastName()
31 {
32 return lastName;
33 } // fim do método getLastName
34
35 // método estático para obter o valor da variável static count
36 public static int getCount()
37 {
38 return count;
39 } // fim do método getCount
40 } // fim da classe Employee

```

**Figura F.12** Variável static usada para manter a contagem do número de objetos Employee na memória.

Quando existem objetos Employee, a variável count pode ser usada em qualquer método de um objeto Employee – este exemplo incrementa count no construtor (linha 18). O método public static getCount (linhas 36 a 39) retorna o número de objetos Employee criados até o momento. Quando não existe objeto da classe Employee, o código cliente pode acessar a variável count chamando o método getCount por meio do nome da classe, como em Employee.getCount(). Quando existem objetos, o método getCount também pode ser chamado por meio de qualquer referência a um objeto Employee.



### Boa prática de programação F.1

*Chame métodos estáticos usando o nome da classe e um ponto (.) para enfatizar que o método que está sendo chamado é static.*

O método main de EmployeeTest (Fig. F.13) instancia dois objetos Employee (linhas 13 e 14). Quando o construtor de cada objeto Employee é chamado, as linhas 15 e 16 da Fig. F.12 atribuem o nome e o sobrenome do funcionário às variáveis de instância firstName e lastName. Essas duas instruções *não* fazem cópias dos argumentos String originais. Na verdade, em Java os objetos String são **imutáveis** – depois de criados, eles não podem ser modificados. Portanto, é seguro ter muitas referências a um único objeto String. Isso normalmente não acontece para objetos da maioria das outras classes em Java. Se os objetos String são imutáveis, talvez você esteja pensando por que podemos usar os operadores + e += para concatená-los. As operações de concatenação de String resultam em um *novo* objeto String, contendo os valores concatenados. Os objetos String originais não são modificados.

Quando main termina de usar os dois objetos Employee, as referências e1 e e2 são configuradas como null nas linhas 31 e 32 (Fig. F.13). Nesse ponto, as referências e1 e e2 não se referem mais aos objetos instanciados nas linhas 13 e 14. Os objetos se tornam “qualificados para a coleta de lixo”, pois não existem mais referências para eles no programa.

```

1 // Fig. F.13: EmployeeTest.java
2 // Demonstração de membro estático.
3
4 public class EmployeeTest
5 {
6 public static void main(String[] args)
7 {
8 // mostra que count é 0 antes de criar objetos Employee
9 System.out.printf("Employees before instantiation: %d\n",
10 Employee.getCount());
11
12 // cria dois objetos Employee; count deve ser 2
13 Employee e1 = new Employee("Susan", "Baker");
14 Employee e2 = new Employee("Bob", "Blue");
15
16 // mostra que count é 2 após a criação dos dois objetos Employee
17 System.out.println("\nEmployees after instantiation: ");
18 System.out.printf("via e1.getCount(): %d\n", e1.getCount());
19 System.out.printf("via e2.getCount(): %d\n", e2.getCount());
20 System.out.printf("via Employee.getCount(): %d\n",
21 Employee.getCount());
22
23 // obtém os nomes dos funcionários
24 System.out.printf("\nEmployee 1: %s %s\nEmployee 2: %s %s\n",
25 e1.getFirstName(), e1.getLastName(),
26 e2.getFirstName(), e2.getLastName());
27
28 // neste exemplo, há apenas uma referência para cada Employee;
29 // portanto, as duas instruções a seguir indicam que esses objetos
30 // estão qualificados para a coleta de lixo
31 e1 = null;
32 e2 = null;
33 } // fim de main
34 } // fim da classe EmployeeTest

```

```

Employees before instantiation: 0
Employee constructor: Susan Baker; count = 1
Employee constructor: Bob Blue; count = 2

Employees after instantiation:
via e1.getCount(): 2
via e2.getCount(): 2
via Employee.getCount(): 2

Employee 1: Susan Baker
Employee 2: Bob Blue

```

**Figura F.13** Demonstração de membro estático.

Finalmente o coletor de lixo poderá recuperar a memória desses objetos (ou o sistema operacional recuperará, quando o programa terminar). A JVM não garante quando, ou mesmo se, o coletor de lixo será executado. Quando ela fizer isso, é possível que nenhum objeto ou apenas um subconjunto dos objetos qualificados seja coletado.

## F.11 Variáveis de instância final

O **princípio do privilégio mínimo** é fundamental para a boa engenharia de software. No contexto de um aplicativo, ele diz que, ao código, deve ser garantida apenas a quantidade necessária de privilégio e acesso para executar sua tarefa designada, não mais que

isso. Isso torna seus programas mais robustos, evitando que código modifique valores de variável acidentalmente (ou de forma mal-intencionada) e chame métodos que não devem ser acessíveis.

Vamos ver como esse princípio se aplica às variáveis de instância. Algumas delas precisam ser modificáveis e outras não. Use a palavra-chave `final` para especificar que uma variável não é modificável (isto é, é uma constante) e que qualquer tentativa de modificá-la é um erro. Por exemplo,

```
private final int INCREMENT;
```

declara uma variável de instância `final` (constante) `INCREMENT` de tipo `int`. Tais variáveis podem ser inicializadas ao serem declaradas. Se não forem, então *devem* ser inicializadas em cada construtor da classe. Inicializar constantes nos construtores permite que cada objeto da classe tenha um valor diferente para a constante. Se uma variável `final` não é inicializada em sua declaração em cada construtor, ocorre um erro de compilação.



#### Observação sobre engenharia de software F.6

*Declarar uma variável de instância como `final` ajuda a impor o princípio do privilégio mínimo. Se uma variável de instância não deve ser modificada, declare-a como `final` para impedir sua modificação.*



#### Erro de programação comum F.7

*Tentar modificar uma variável de instância `final` depois que ela for inicializada é um erro de compilação.*



#### Dica de prevenção de erro F.2

*As tentativas de modificar uma variável de instância `final` são capturadas no momento da compilação, em vez de causar erros de tempo de execução. É sempre preferível eliminar erros no momento da compilação, se possível, em vez de permitir que apareçam no momento da execução (a experiência tem mostrado que, frequentemente, o reparo é muito mais dispendioso quando ocorre no momento da execução).*



#### Observação sobre engenharia de software F.7

*Um campo `final` também deve ser declarado como `static`, caso seja inicializado em sua declaração com um valor igual para todos os objetos da classe. Depois dessa inicialização, seu valor nunca pode mudar. Portanto, não precisamos de uma cópia separada do campo para cada objeto da classe. Tornar o campo estático permite que todos os objetos da classe compartilhem esse campo.*

## F.12 Pacotes

Em quase todos os exemplos que estudamos, classes de bibliotecas já existentes, como a API Java, podiam ser importadas para um programa Java. Cada classe da API Java pertence a um pacote que contém um grupo de classes relacionadas. Esses pacotes são definidos apenas uma vez, mas podem ser importados em muitos programas. À medida que os aplicativos se tornam mais complicados, os pacotes ajudam a gerenciar a complexidade dos seus componentes. Os pacotes também facilitam a reutilização de software, permitindo que os programas *importem* classes de outros pacotes (como fizemos na maioria

dos exemplos), em vez de *copiá-las* em cada programa que as utilize. Outra vantagem dos pacotes é que eles fornecem uma convenção para nomes de classe exclusivos, o que ajuda a evitar conflitos de nomes de classe.

## F.13 Acesso ao pacote

Se nenhum modificador de acesso (`public`, `protected` ou `private`) for especificado para um método ou variável quando da sua declaração em uma classe, o método ou variável tem **acesso ao pacote**. Em um programa composto de uma única declaração de classe, isso não tem um efeito específico. No entanto, se um programa usa várias classes do mesmo pacote (isto é, um grupo de classes relacionadas), essas classes podem acessar diretamente os membros de acesso a pacote uma das outras por meio de referências para objetos das classes apropriadas, ou, no caso de membros estáticos, por meio do nome da classe. O acesso ao pacote é raramente usado.

## F.14 Para finalizar

Neste apêndice, apresentamos mais conceitos sobre classes. O estudo de caso da classe `Time` apresentou uma declaração de classe completa, consistindo em dados `private`, construtores `public` sobrecarregados para proporcionar flexibilidade na inicialização, métodos `set` e `get` para manipular os dados da classe e métodos que retornavam representações de `String` de um objeto `Time` em dois formatos diferentes. Você aprendeu também que toda classe pode declarar um método `toString` que retorna uma representação de `String` de um objeto da classe e que o método `toString` pode ser chamado implicitamente sempre que um objeto de uma classe aparece no código, quando é esperada uma `String`.

Você aprendeu que a referência `this` é usada implicitamente nos métodos não estáticos de uma classe para acessar as variáveis de instância da classe e outros métodos não estáticos. Viu também usos explícitos da referência `this` para acessar os membros da classe (incluindo campos sombreados) e como usar a palavra-chave `this` em um construtor para chamar outro construtor da classe.

Discutimos as diferenças entre os construtores padrão fornecidos pelo compilador e os construtores sem argumentos fornecidos pelo programador. Você aprendeu que uma classe pode ter referências para objetos de outras classes como membros – um conceito conhecido como composição. Viu também o tipo de classe `enum` e aprendeu como pode ser empregado para criar um conjunto de constantes para uso em um programa. Você conheceu a capacidade de coleta de lixo do Java e como ela recupera (de forma imprevisível) a memória dos objetos que não são mais utilizados. Explicamos a motivação para usar campos estáticos em uma classe e demonstramos como declarar e usar campos e métodos estáticos em suas próprias classes. Você aprendeu também a declarar e inicializar variáveis `final`.

Você aprendeu que os campos declarados sem um modificador de acesso recebem acesso ao pacote por padrão e que as classes do mesmo pacote podem acessar os membros com acesso ao pacote de outras classes do pacote.

No próximo apêndice, você vai aprender sobre dois importantes aspectos da programação orientada a objetos com Java – herança e polimorfismo. Você vai ver que todas as classes em Java estão relacionadas direta ou indiretamente com a classe chamada `Object`. Também vai começar a entender como as relações entre as classes permitem construir aplicativos mais poderosos.

## Exercício de revisão

- F.1** Preencha os espaços em branco em cada um dos seguintes enunciados:
- Os métodos `public` de uma classe também são conhecidos como \_\_\_\_\_ ou \_\_\_\_\_ da classe.
  - O método \_\_\_\_\_ da classe `String` é semelhante ao método `System.out.printf`, mas retorna uma `String` formatada, em vez de exibir uma `String` em uma janela de comando.
  - Se um método contém uma variável local com o mesmo nome de um dos campos de sua classe, a variável local \_\_\_\_\_ o campo no escopo desse método.
  - A palavra-chave \_\_\_\_\_ especifica que uma variável não é modificável.
  - O \_\_\_\_\_ diz que deve ser garantida ao código apenas a quantidade necessária de privilégio e acesso para executar sua tarefa designada.
  - Se uma classe declara construtores, o compilador não criará um \_\_\_\_\_.
  - O método \_\_\_\_\_ de um objeto é chamado implicitamente quando o objeto aparece no código em que uma `String` é necessária.
  - Para cada `enum`, o compilador gera um método estático chamado \_\_\_\_\_ que retorna um array das constantes de `enum` na ordem em que foram declaradas.
  - A composição às vezes é referida como relação \_\_\_\_\_.
  - Uma declaração \_\_\_\_\_ contém uma lista de constantes separadas por vírgulas.
  - Uma variável \_\_\_\_\_ representa informações em nível de classe que não são compartilhadas por todos os objetos da classe.

## Respostas do exercício de revisão

- F.1** a) serviços `public`, interface `public`. b) `format`. c) `sombrea`. d) `final`. e) princípio do privilégio mínimo. f) construtor padrão. g) `toString`. h) `values`. i) *tem um*. j) `enum`. k) estática.

## Exercícios

- F.2** (*Classe `Rectangle`*) Crie uma classe `Rectangle` com atributos `length` e `width` (comprimento e largura), cada um dos quais tendo 1 como padrão. Forneça métodos que calculem o perímetro e a área do retângulo. Forneça métodos *set* e *get* para `length` e `width`. Os métodos *set* devem verificar se `length` e `width` são números de ponto flutuante maiores que 0.0 e menores que 20.0. Escreva um programa para testar a classe `Rectangle`.
- F.3** (*Classe `SavingsAccount` - Poupança*) Crie a classe `SavingsAccount`. Use uma variável estática `annualInterestRate` para armazenar a taxa de juros anual de todos os correntistas. Cada objeto da classe deve conter um variável de instância `savingsBalance` indicando o valor que o poupador tem atualmente depositado. Forneça o método `calculateMonthlyInterest` para calcular os juros mensais, multiplicando `savingsBalance` por `annualInterestRate` dividido por 12 – esses juros devem ser somados a `savingsBalance`. Forneça um método estático `modifyInterestRate` que configure `annualInterestRate` com um novo valor. Escreva um programa para testar a classe `SavingsAccount`. Instancie dois objetos `savingsAccount`, `saver1` e `saver2`, com saldos de US\$2000.00 e US\$3000.00, respectivamente. Configure `annualInterestRate` como 4% e então calcule os juros mensais para cada um dos 12 meses e imprima os novos saldos dos dois poupadores. Em seguida, configure `annualInterestRate` como 5%, calcule os juros do próximo mês e imprima os novos saldos dos dois poupadores.
- F.4** (*Melhorando a classe `Time2`*) Modifique a classe `Time2` da Fig. F.5 para incluir um método `tick` que incremente por segundo a hora armazenada em um objeto `Time2`. Forneça o



método `incrementMinute` para incrementar o minuto por 1 e o método `incrementHour` para incrementar a hora por 1. Escreva um programa que teste o método `tick`, o método `incrementMinute` e o método `incrementHour` a fim de garantir que eles funcionam corretamente. Certifique-se de testar os seguintes casos:

- a) incrementar pelo próximo minuto
- b) incrementar pela próxima hora
- c) incrementar pelo próximo dia (isto é, de 11:59:59 PM a 12:00:00 AM).

**F.5** Escreva um tipo `enum TrafficLight`, cujas constantes (`RED`, `GREEN`, `YELLOW`) recebam um único parâmetro – a duração do sinal. Escreva um programa para testar o `enum TrafficLight` de modo que ele exiba as constantes `enum` e suas durações.

**F.6** (*Classe Date*) Crie a classe `Date` com os seguintes recursos:

- a) Gere a data em vários formatos, como

```
MM/DD/AAAA
Junho 14, 1992
DDD AAAA
```

- b) Use construtores sobrecarregados para criar objetos `Date` inicializados com datas nos formatos da parte (a). No primeiro caso, o construtor deve receber três valores inteiros. No segundo, deve receber uma `String` e dois valores inteiros. No terceiro, deve receber dois valores inteiros, o primeiro dos quais representando o número do dia no ano. [*Dica:* para converter a representação de `String` do mês em um valor numérico, compare `Strings` usando o método `equals`. Por exemplo, se `s1` e `s2` são `Strings`, a chamada de método `s1.equals( s2 )` retorna `true` se as `Strings` são idênticas e, caso contrário, retorna `false`.]

**F.7** (*Classe HugeInteger*) Crie uma classe `HugeInteger` que utilize um array de dígitos de 40 elementos para armazenar valores inteiros de até 40 dígitos cada um. Forneça os métodos `parse`, `toString`, `add` e `subtract`. O método `parse` deve receber uma `String`, extrair cada dígito usando o método `charAt` e colocar o inteiro equivalente de cada dígito no array inteiro. Para comparar objetos `HugeInteger`, forneça os seguintes métodos: `isEqualTo`, `isNotEqualTo`, `isGreaterThan`, `isLessThan`, `isGreaterThanOrEqualTo` e `isLessThanOrEqualTo`. Cada um desses assim chamados *métodos de predicado* (isto é, métodos que testam uma condição e retornam `true` ou `false`) retorna `true` se a relação vale entre os dois objetos `HugeInteger` e `false` se a relação não vale. Forneça um método de predicado `isZero`. Caso queira ir um pouco além, forneça também os métodos `multiply`, `divide` e `remainder`. [*Obs.:* os valores primitivos `boolean` podem ser gerados como a palavra “true” ou a palavra “false” com o especificador de formato `%b`.]

**F.8** (*Tic-Tac-Toe - Jogo da Velha*) Crie uma classe `TicTacToe` (jogo da velha) que permita escrever um programa para jogar o Tic-Tac-Toe (Jogo da Velha). A classe contém um array bidimensional `private` de 3 por 3. Use uma enumeração para representar o valor de cada célula do array. As constantes da enumeração devem se chamar `X`, `O` e `EMPTY` (para uma posição que não contém `X` nem `O`). O construtor deve inicializar os elementos do tabuleiro como `EMPTY` (vazio). Permita que duas pessoas joguem. Quando for a vez do primeiro jogador, coloque um `X` no quadrado especificado e, quando o outro jogar, coloque um `O`. Cada jogada deve acontecer em um quadrado vazio. Após cada jogada, determine se o jogo foi vencido ou se houve um empate. Se você quiser ir um pouco além, modifique seu programa de modo que o computador seja um dos jogadores. Além disso, permita que o jogador especifique se deseja ser o primeiro ou o segundo a jogar. Se você quiser ir ainda mais além, desenvolva um programa que jogue um Tic-Tac-Toe (Jogo da Velha) tridimensional em um tabuleiro de 4 por 4 por 4. [*Obs.:* esse é um projeto extremamente desafiador!].

# Programação orientada a objetos: herança e polimorfismo

# G

## Objetivos

Neste capítulo, você vai:

- Saber como a herança promove a reutilização de software.
- Entender as relações entre superclasses e subclasses.
- Usar a palavra-chave `extends` para produzir herança.
- Usar `protected` para que métodos de subclasses acessem membros de superclasses.
- Fazer referência a membros da superclasse com `super`.
- Conhecer os métodos da classe `Object`.
- Aprender o conceito do polimorfismo.
- Usar métodos sobrescritos para produzir polimorfismo.
- Distinguir entre classes abstratas e concretas.
- Declarar métodos abstratos para criar classes abstratas.
- Saber como o polimorfismo torna os sistemas extensíveis e fáceis de manter.
- Determinar o tipo de um objeto no momento da execução.
- Declarar e implementar interfaces.



- |                                                      |                                                                               |
|------------------------------------------------------|-------------------------------------------------------------------------------|
| <b>G.1</b> Introdução à herança                      | <b>G.9</b> Classes e métodos abstratos                                        |
| <b>G.2</b> Superclasses e subclasses                 | <b>G.10</b> Estudo de caso: sistema de folha de pagamento usando polimorfismo |
| <b>G.3</b> Membros <code>protected</code>            | <b>G.11</b> Métodos e classes <code>final</code>                              |
| <b>G.4</b> Relações entre superclasses e subclasses  | <b>G.12</b> Estudo de caso: criação e uso de interfaces                       |
| <b>G.5</b> Classe <code>Object</code>                | <b>G.13</b> Interfaces comuns da API Java                                     |
| <b>G.6</b> Introdução ao polimorfismo                | <b>G.14</b> Para finalizar                                                    |
| <b>G.7</b> Polimorfismo: um exemplo                  |                                                                               |
| <b>G.8</b> Demonstração de comportamento polimórfico |                                                                               |

*Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios*

## G.1 Introdução à herança

A primeira parte deste apêndice continua nossa discussão sobre a programação orientada a objetos (POO), apresentando um de seus principais recursos: a **herança** – uma forma de reutilização de software na qual uma nova classe é criada absorvendo os membros de uma classe já existente e complementando-os com recursos novos ou modificados. Com a herança, você pode economizar tempo durante o desenvolvimento de programas, baseando novas classes em software já existente de alta qualidade, testado e depurado. A classe existente é denominada **superclasse** e a nova classe é a **subclasse**. Cada subclasse pode se tornar uma superclasse de futuras subclasses.

Uma subclasse pode adicionar seus próprios campos e métodos. Portanto, uma subclasse é *mais específica* do que sua superclasse e representa um grupo de objetos mais especializado. A subclasse exibe os comportamentos de sua superclasse e pode modificá-los para que funcionem adequadamente na subclasse. É por isso que a herança às vezes é referida como **especialização**.

A **superclasse direta** é aquela da qual a subclasse herda explicitamente. Uma **superclasse indireta** é qualquer classe acima da superclasse direta na **hierarquia de classes**, a qual define as relações de herança entre as classes. Em Java, a hierarquia de classes começa com a classe `Object` (no pacote `java.lang`), a qual *toda* classe Java **estende** (ou “herda de”) direta ou indiretamente. A seção G.5 lista os métodos da classe `Object` herdados por todas as outras classes Java.

Fazemos diferenciação entre a **relação é um** e a **relação tem um**. *É um* representa a herança. Em uma relação *é um*, *um objeto de uma subclasse também pode ser tratado como um objeto de sua superclasse* – por exemplo, um carro *é um* veículo. Em contraste, *tem um* representa a composição (consulte o Apêndice F). Em uma relação *tem um*, *um objeto contém como membros referências para outros objetos* – por exemplo, um carro *tem um* volante (e um objeto carro tem uma referência para um objeto volante).

Mais adiante neste apêndice, discutimos a noção de polimorfismo, o qual simplifica a programação com objetos da mesma hierarquia de classes. Você vai ver que o polimorfismo também torna possível estender sistemas com a adição de novos recursos. Por fim, discutimos as interfaces, as quais são úteis para atribuir funcionalidade comum a classes possivelmente *não relacionadas*. Isso permite que objetos de classes não relacionadas sejam processados de modo polimórfico – objetos de classes que implementam a mesma interface podem responder a todas as chamadas de métodos da interface de forma personalizada.

## G.2 Superclasses e subclasses

Frequentemente, um objeto de uma classe também *é um* objeto de outra classe. A Figura G.1 lista vários exemplos de superclasses e subclasses – as superclasses tendem a ser “mais gerais” e as subclasses, “mais específicas”. Por exemplo, *EmpréstimoDeCarro é um Empréstimo*, assim como *EmpréstimoParaReforma* e *EmpréstimoHipotecário*. Assim, em Java, pode-se dizer que a classe *EmpréstimoDeCarro* herda da classe *Empréstimo*. Nesse contexto, a classe *Empréstimo* é uma superclasse e classe *EmpréstimoDeCarro* é uma subclasse. *EmpréstimoDeCarro é um* tipo específico de *Empréstimo*, mas é incorreto dizer que todo *Empréstimo é um EmpréstimoDeCarro* – *Empréstimo* poderia ser qualquer tipo de empréstimo.

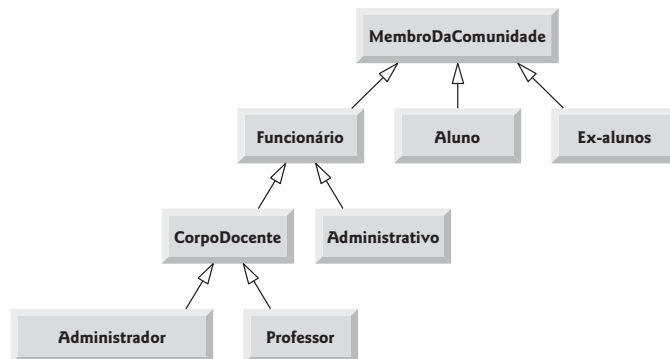
| Superclasse   | Subclasses                                                      |
|---------------|-----------------------------------------------------------------|
| Aluno         | AlunoGraduado, AlunoNãoGraduado                                 |
| Forma         | Círculo, Triângulo, Retângulo, Esfera, Cubo                     |
| Empréstimo    | EmpréstimoDeCarro, EmpréstimoParaReforma, EmpréstimoHipotecário |
| Funcionário   | Docente, Administrativo                                         |
| ContaBancária | ContaCorrente, Poupança                                         |

**Figura G.1** Exemplos de herança.

Como todo objeto de subclasse *é um* objeto de sua superclasse e como uma superclasse pode ter muitas subclasses, frequentemente o conjunto de objetos representados por uma superclasse é maior que o conjunto de objetos representados por qualquer uma de suas subclasses. Por exemplo, a superclasse *Veículo* representa todos os veículos, incluindo carros, caminhões, barcos, bicicletas, etc. Em contraste, a subclasse *Carro* representa um subconjunto menor e mais específico de veículos.

### Hierarquia de membros da comunidade universitária

Os relacionamentos de herança formam estruturas hierárquicas do tipo árvore. Há uma relação hierárquica entre a superclasse e suas subclasses. Vamos desenvolver um exemplo de hierarquia de classes (Fig. G.2), também chamada de **hierarquia de herança**. Uma comunidade universitária tem milhares de membros, incluindo funcionários, alunos e ex-alunos. Os funcionários são membros do corpo docente ou administrativos.



**Figura G.2** Hierarquia de herança para MembroDaComunidade.

Os membros do corpo docente são administradores (por exemplo, reitores e chefes de departamento) ou professores. A hierarquia poderia conter muitas outras classes. Por exemplo, alunos podem ser graduados ou não graduados. Os não graduados podem ser calouros, alunos do segundo ano, do penúltimo ano ou do último ano.

Cada seta na hierarquia representa uma relação *é um*. Quando seguimos as setas para cima nessa hierarquia de classes, podemos dizer, por exemplo, que “um Funcionário *é um* MembroDaComunidade” e que “um Professor *é um* membro do CorpoDocente”. MembroDaComunidade é a superclasse direta de Funcionário, Aluno e Ex-alunos, e é uma superclasse indireta de todas as outras classes no diagrama. De baixo para cima, você pode seguir as setas e aplicar a relação *é um* até a superclasse superior. Por exemplo, um Administrador *é um* membro do CorpoDocente, *é um* Funcionário, *é um* MembroDaComunidade e, evidentemente, *é um* Object.

### Hierarquia de Formas

Agora, considere a hierarquia de herança Forma da Fig. G.3. Essa hierarquia começa com a superclasse Forma, a qual é estendida pelas subclasses FormaBidimensional e FormaTridimensional — as formas são bidimensionais ou tridimensionais. O terceiro nível dessa hierarquia contém tipos específicos de formas bidimensionais e tridimensionais. Como na Fig. G.2, podemos seguir as setas de baixo para cima no diagrama, até a superclasse superior dessa hierarquia de classes, para identificar várias relações *é um*. Por exemplo, um Triângulo *é uma* FormaBidimensional e *é uma* Forma, enquanto uma Esfera *é uma* FormaTridimensional e *é uma* Forma. Essa hierarquia poderia conter muitas outras classes. Por exemplo, ovais e trapezoides são FormasBidimensionais.

É possível tratar objetos de superclasse e objetos de subclasse de forma similar — suas características comuns são expressas nos membros da superclasse. Objetos de todas as classes que estendem uma superclasse comum podem ser tratados como objetos dessa superclasse — tais objetos têm uma relação *é um* com a superclasse. Mais adiante neste apêndice, consideramos muitos exemplos que tiram proveito da relação *é um*.

Uma subclasse pode personalizar os métodos que herda de sua superclasse. Para isso, a subclasse **sobrescreve** (redefine) o método da superclasse com uma implementação apropriada, como veremos muitas vezes nos exemplos de código deste apêndice.

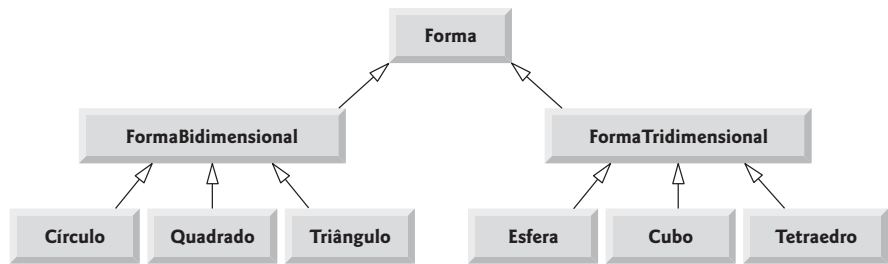


Figura G.3 Hierarquia de herança para Formas.

## G.3 Membros protected

Nesta seção, apresentamos o modificador de acesso **protected**. O uso de **protected** oferece um nível intermediário de acesso entre **public** e **private**. Os membros **protected** de uma superclasse podem ser acessados pela classe, pelos membros de suas subclasses

e pelos membros de outras classes do mesmo pacote – os membros `protected` também têm acesso ao pacote.

Todos os membros `public` e `protected` da superclasse mantêm seus modificadores de acesso originais ao se tornarem membros da subclasse – os membros `public` da superclasse se tornam membros `public` da subclasse e os membros `protected` da superclasse se tornam membros `protected` da subclasse. Os membros `private` da superclasse *não* são acessíveis fora da própria classe. Em vez disso, ficam *ocultos* em suas subclasses e só podem ser acessados por meio dos métodos `public` ou `protected` herdados da superclasse.

Os métodos da subclasse podem fazer referência aos membros `public` e `protected` herdados da superclasse simplesmente usando os nomes dos membros. Quando um método de uma subclasse sobrescreve um método herdado da superclasse, o método da *superclasse* pode ser acessado a partir da *subclasse*, precedendo-se o nome do método da superclasse com a palavra-chave **super** e um separador ponto (`.`). Discutimos o acesso a membros sobrescritos da superclasse na seção G.4.

## G.4 Relações entre superclasses e subclasses

Usamos agora uma hierarquia de herança contendo tipos de funcionários em um aplicativo de folha de pagamento de uma empresa para discutirmos a relação entre uma superclasse e sua subclasse. Nessa empresa, os funcionários comissionados (que serão representados como objetos de uma superclasse) recebem uma porcentagem de suas vendas, enquanto os funcionários assalariados (que serão representados como objetos de uma subclasse) recebem um salário-base *mais* uma porcentagem de suas vendas. Criamos um exemplo que configura as variáveis de instância `CommissionEmployee` (para funcionários comissionados) como `private` para impor a boa engenharia de software. Então, mostramos como a subclasse `BasePlusCommissionEmployee` (para funcionários assalariados) pode usar métodos `public` de `CommissionEmployee` para manipular (de forma controlada) as variáveis de instância `private` herdadas de `CommissionEmployee`.

### G.4.1 Criação e uso de uma classe `CommissionEmployee`

Começamos declarando a classe `CommissionEmployee` (Fig. G.4). A linha 4 inicia a declaração da classe e indica que a classe `CommissionEmployee` estende (palavra-chave **extends** – isto é, herda da) a classe `Object` (do pacote `java.lang`). Isso faz a classe `CommissionEmployee` herdar os métodos da classe `Object` – a classe `Object` não tem campos. Se você não especifica explicitamente qual classe uma nova classe estende, ela estende `Object` implicitamente. Por isso, normalmente não se inclui “`extends Object`” no código – fizemos isso neste exemplo somente para propósitos de demonstração.

#### **Visão geral dos métodos e variáveis de instância da classe `CommissionEmployee`**

Os serviços `public` da classe `CommissionEmployee` incluem um construtor (linhas 13 a 22) e os métodos `earnings` (linhas 93 a 96) e `toString` (linhas 99 a 107). As linhas 25 a 90 declaram métodos `public` `get` e `set` para as variáveis de instância `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` e `commissionRate` (declaradas nas linhas 6 a 10) da classe. A classe declara suas variáveis de instância como `private`, de modo que objetos de outras classes não podem acessá-las diretamente. Declarar variáveis de instância como `private` e fornecer métodos `get` e `set` para manipulá-las e validá-las ajuda a impor a boa engenharia de software. Os métodos `setGrossSales` e `setCommissionRate`, por exemplo,

validam seus argumentos antes de atribuir os valores para as variáveis de instância grossSales e commissionRate, respectivamente. Em um aplicativo real, fundamental para uma empresa, também faríamos a validação nos outros métodos *set* da classe.

```

1 // Fig. G.4: CommissionEmployee.java
2 // A classe CommissionEmployee representa um funcionário pago
3 // por uma porcentagem das vendas brutas.
4 public class CommissionEmployee extends Object
5 {
6 private String firstName;
7 private String lastName;
8 private String socialSecurityNumber;
9 private double grossSales; // vendas brutas semanais
10 private double commissionRate; // porcentagem de comissão
11
12 // construtor de cinco argumentos
13 public CommissionEmployee(String first, String last, String ssn,
14 double sales, double rate)
15 {
16 // a chamada implícita para o construtor de Object ocorre aqui
17 firstName = first;
18 lastName = last;
19 socialSecurityNumber = ssn;
20 setGrossSales(sales); // valida e armazena as vendas brutas
21 setCommissionRate(rate); // valida e armazena a taxa de comissão
22 } // fim do construtor de cinco argumentos de CommissionEmployee
23
24 // configura o nome
25 public void setFirstName(String first)
26 {
27 firstName = first; // deve validar
28 } // fim do método setFirstName
29
30 // retorna o nome
31 public String getFirstName()
32 {
33 return firstName;
34 } // fim do método getFirstName
35
36 // configura o sobrenome
37 public void setLastName(String last)
38 {
39 lastName = last; // deve validar
40 } // fim do método setLastName
41
42 // retorna o sobrenome
43 public String getLastName()
44 {
45 return lastName;
46 } // fim do método getLastName
47
48 // configura o número da previdência social
49 public void setSocialSecurityNumber(String ssn)
50 {
51 socialSecurityNumber = ssn; // deve validar
52 } // fim do método setSocialSecurityNumber
53
54 // retorna o número da previdência social
55 public String getSocialSecurityNumber()

```

**Figura G.4** A classe *CommissionEmployee* representa um funcionário pago por uma porcentagem das vendas brutas. (continua)

```

56 {
57 return socialSecurityNumber;
58 } // fim do método getSocialSecurityNumber
59
60 // configura o valor das vendas brutas
61 public void setGrossSales(double sales)
62 {
63 if (sales >= 0.0)
64 grossSales = sales;
65 else
66 throw new IllegalArgumentException(
67 "Gross sales must be >= 0.0");
68 } // fim do método setGrossSales
69
70 // retorna o valor das vendas brutas
71 public double getGrossSales()
72 {
73 return grossSales;
74 } // fim do método getGrossSales
75
76 // configura a taxa de comissão
77 public void setCommissionRate(double rate)
78 {
79 if (rate > 0.0 && rate < 1.0)
80 commissionRate = rate;
81 else
82 throw new IllegalArgumentException(
83 "Commission rate must be > 0.0 and < 1.0");
84 } // fim do método setCommissionRate
85
86 // retorna a taxa de comissão
87 public double getCommissionRate()
88 {
89 return commissionRate;
90 } // fim do método getCommissionRate
91
92 // calcula os ganhos
93 public double earnings()
94 {
95 return commissionRate * grossSales;
96 } // fim do método earnings
97
98 // retorna representação de String do objeto CommissionEmployee
99 @Override // indica que esse método sobreescreve um método da superclasse
100 public String toString()
101 {
102 return String.format("%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f",
103 "commission employee", firstName, lastName,
104 "social security number", socialSecurityNumber,
105 "gross sales", grossSales,
106 "commission rate", commissionRate);
107 } // fim do método toString
108 } // fim da classe CommissionEmployee

```

**Figura G.4** A classe `CommissionEmployee` representa um funcionário pago por uma porcentagem das vendas brutas.

### Construtor da classe *CommissionEmployee*

Construtores *não* são herdados; portanto, a classe `CommissionEmployee` não herda o construtor da classe `Object`. Contudo, os construtores de uma superclasse ainda estão disponíveis para as subclasses. Na verdade, *a primeira tarefa do construtor de qualquer subclasse é*



*chamar o construtor de sua superclasse direta*, ou explicitamente ou implicitamente (se nenhuma chamada de construtor for especificada), para garantir que as variáveis de instância herdadas da superclasse sejam inicializadas corretamente. Neste exemplo, o construtor da classe `CommissionEmployee` chama o construtor da classe `Object` implicitamente. A sintaxe para chamar o construtor de uma superclasse explicitamente está discutida na seção G.4.3. Se o código não inclui uma chamada explícita ao construtor da superclasse, a linguagem Java chama *implicitamente* o construtor padrão ou o construtor sem argumentos da superclasse. O comentário na linha 16 da Fig. G.4 indica onde é feita a chamada implícita ao construtor padrão da superclasse `Object` (você não escreve o código para essa chamada). O construtor padrão (vazio) de `Object` não faz nada. Mesmo que uma classe não tenha construtores, o construtor padrão declarado implicitamente pelo compilador para a classe chamará o construtor padrão ou o construtor sem argumentos da superclasse.

Após a chamada implícita ao construtor de `Object`, as linhas 17 a 21 do construtor de `CommissionEmployee` atribuem valores para as variáveis de instância da classe. Não validamos os valores dos argumentos `first`, `last` e `ssn` antes de atribuí-los às variáveis de instância correspondentes. Poderíamos validar os nomes e sobrenomes – talvez para garantir que tenham um comprimento razoável. Do mesmo modo, um número de previdência social poderia ser validado com expressões regulares para garantir que contivessem nove algarismos, com ou sem traços (por exemplo, 123-45-6789 ou 123456789).

### **Método `earnings` da classe `CommissionEmployee`**

O método `earnings` (linhas 93 a 96) calcula os ganhos de um funcionário comissionado (`CommissionEmployee`). A linha 95 multiplica `commissionRate` por `grossSales` e retorna o resultado.

### **Método `toString` da classe `CommissionEmployee` e a anotação `@Override`**

O método `toString` (linhas 99 a 107) é especial – ele é um dos métodos que *toda* classe herda direta ou indiretamente da classe `Object` (resumida na seção G.5). O método `toString` retorna uma `String` representando um objeto. Ele é chamado implicitamente sempre que um objeto precisa ser convertido em uma representação de `String`, como quando um objeto é gerado na saída por `printf` ou pelo método `format` de `String`, por meio do especificador de formato `%s`. O método `toString` da classe `Object` retorna uma `String` que inclui o nome da classe do objeto. Ele é principalmente um espaço reservado que pode ser sobrescrito por uma subclasse para especificar uma representação de `String` apropriada dos dados de um objeto de uma subclasse. O método `toString` da classe `CommissionEmployee` sobrescreve (redefine) o método `toString` da classe `Object`. Quando chamado, o método `toString` de `CommissionEmployee` utiliza o método `format` de `String` para retornar uma `String` contendo informações sobre o `CommissionEmployee`. Para sobrescrever um método de uma superclasse, uma subclasse deve declarar um método com a mesma assinatura (nome do método, número de parâmetros, tipos de parâmetro e ordem dos tipos de parâmetro) do método da superclasse – o método `toString` de `Object` não recebe parâmetros; portanto, `CommissionEmployee` declara `toString` sem parâmetros.

A linha 99 usa a **anotação `@Override`** para indicar que o método `toString` deve sobrescrever um método de superclasse. As anotações têm várias finalidades. Por exemplo, quando você tenta sobrescrever um método de superclasse, erros comuns incluem dar nome incorreto ao método da subclasse ou usar o número ou os tipos errados de parâmetros na lista de parâmetros. Cada um desses problemas gera uma *sobrecarga involuntária* do método da superclasse. Se você tentar, então, chamar o método em um objeto

da subclasse, a versão da superclasse será chamada e a da subclasse será ignorada – talvez levando a erros de lógica sutis. Quando o compilador encontra um método declarado com `@Override`, ele compara a assinatura do método com as assinaturas de método da superclasse. Se não há uma correspondência exata, o compilador gera uma mensagem de erro, como “method does not override or implement a method from a supertype.” Isso indica que você sobrecarregou acidentalmente um método da superclasse. Você pode então corrigir a assinatura de seu método para que corresponda ao da superclasse.

Em aplicativos web e web services, as anotações também podem adicionar código de suporte complexo a suas classes para simplificar o processo de desenvolvimento e ser usado por servidores para configurar certos aspectos dos aplicativos web.



### Erro de programação comum G.1

*É um erro de sintaxe sobrepor um método com um modificador de acesso mais restrito – um método `public` da superclasse não pode se tornar um método `protected` ou `private` na subclasse; um método `protected` da superclasse não pode se tornar um método `private` na subclasse. Isso violaria a relação de um, pois é obrigatório todos os objetos da subclasse responderem às chamadas de método feitas para métodos `public` declarados na superclasse. Se um método `public`, por exemplo, pudesse ser sobrescrito como um método `protected` ou `private`, os objetos da subclasse não poderiam responder às mesmas chamadas de método que os objetos da superclasse. Quando um método é declarado `public` em uma superclasse, ele permanece `public` para todas as subclasses diretas e indiretas dessa classe.*

### Classe `CommissionEmployeeTest`

A Figura G.5 testa a classe `CommissionEmployee`. As linhas 9 e 10 instanciam um objeto `CommissionEmployee` e chamam o construtor de `CommissionEmployee` (linhas 13 a 22 da Fig. G.4) para inicializá-lo com o nome “Sue”, o sobrenome “Jones”, o número de previdência social “222-22-2222”, o valor 10000 para as vendas brutas e a taxa de comissão .06. As linhas 15 a 24 usam os métodos `get` de `CommissionEmployee` para recuperar os valores da variável de instância do objeto para gerar a saída. As linhas 26 e 27 chamam os métodos `setGrossSales` e `setCommissionRate` do objeto para alterar os valores das variáveis de instância `grossSales` e `commissionRate`. As linhas 29 e 30 geram a representação de `String` do objeto `CommissionEmployee` atualizado. Quando um objeto é gerado na saída com o especificador de formato `%s`, seu método `toString` é chamado implicitamente para obter a representação de `String` do objeto. [Obs.: anteriormente neste apêndice, não usamos os métodos `earnings` de nossas classes – eles são muito usados na parte deste apêndice que trata sobre polimorfismo.]

```

1 // Fig. G.5: CommissionEmployeeTest.java
2 // Programa de teste da classe CommissionEmployee.
3
4 public class CommissionEmployeeTest
5 {
6 public static void main(String[] args)
7 {
8 // instancia objeto CommissionEmployee
9 CommissionEmployee employee = new CommissionEmployee(
10 "Sue", "Jones", "222-22-2222", 10000, .06);
11
12 // obtém dados de funcionário comissionado
13 System.out.println(

```

**Figura G.5** Programa de teste da classe `CommissionEmployee`. (continua)

```

14 "Employee information obtained by get methods: \n");
15 System.out.printf("%s %s\n", "First name is",
16 employee.getFirstName());
17 System.out.printf("%s %s\n", "Last name is",
18 employee.getLastName());
19 System.out.printf("%s %s\n", "Social security number is",
20 employee.getSocialSecurityNumber());
21 System.out.printf("%s %.2f\n", "Gross sales is",
22 employee.getGrossSales());
23 System.out.printf("%s %.2f\n", "Commission rate is",
24 employee.getCommissionRate());
25
26 employee.setGrossSales(500); // configura vendas brutas
27 employee.setCommissionRate(.1); // configura taxa de comissão
28
29 System.out.printf("\n%s:\n\n%s\n",
30 "Updated employee information obtained by toString", employee);
31 } // fim de main
32 } // fim da classe CommissionEmployeeTest

```

```

Employee information obtained by get methods:

First name is Sue
Last name is Jones
Social security number is 222-22-2222
Gross sales is 10000.00
Commission rate is 0.06

Updated employee information obtained by toString:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 500.00
commission rate: 0.10

```

**Figura G.5** Programa de teste da classe `CommissionEmployee`.

## G.4.2 Criação e uso de uma classe `BasePlusCommissionEmployee`

Discutimos agora a segunda parte de nossa introdução à herança, declarando e testando a (completamente nova e independente) classe `BasePlusCommissionEmployee` (Fig. G.6), a qual contém nome, sobrenome, número da previdência social, valor das vendas brutas e salário-base. Os serviços públicos da classe `BasePlusCommissionEmployee` incluem um construtor (linhas 15 a 25) e os métodos `earnings` (linhas 112 a 115) e `toString` (linhas 118 a 127). As linhas 28 a 109 declaram métodos públicos *get* e *set* para as variáveis de instância privadas `firstName`, `lastName`, `socialSecurityNumber`, `grossSales`, `commissionRate` e `baseSalary` (declaradas nas linhas 7 a 12) da classe. Essas variáveis e métodos encapsulam todos os recursos necessários para um funcionário assalariado. Note a *semelhança* entre a classe `BasePlusComissionEmployee` e a classe `CommissionEmployee` (Fig. G.4) – neste exemplo, ainda não vamos explorar essa semelhança.

```

1 // Fig. G.6: BasePlusCommissionEmployee.java
2 // A classe BasePlusCommissionEmployee representa um funcionário
3 // que recebe um salário-base mais uma comissão.
4
5 public class BasePlusCommissionEmployee
6 {
7 private String firstName;
8 private String lastName;
9 private String socialSecurityNumber;
10 private double grossSales; // vendas brutas semanais
11 private double commissionRate; // porcentagem de comissão
12 private double baseSalary; // salário-base por semana
13
14 // construtor de seis argumentos
15 public BasePlusCommissionEmployee(String first, String last,
16 String ssn, double sales, double rate, double salary)
17 {
18 // a chamada implícita ao construtor de Object ocorre aqui
19 firstName = first;
20 lastName = last;
21 socialSecurityNumber = ssn;
22 setGrossSales(sales); // valida e armazena as vendas brutas
23 setCommissionRate(rate); // valida e armazena a taxa de comissão
24 setBaseSalary(salary); // valida e armazena o salário-base
25 } // fim do construtor de seis argumentos de BasePlusCommissionEmployee
26
27 // configura o nome
28 public void setFirstName(String first)
29 {
30 firstName = first; // deve validar
31 } // fim do método setFirstName
32
33 // retorna o nome
34 public String getFirstName()
35 {
36 return firstName;
37 } // fim do método getFirstName
38
39 // configura o sobrenome
40 public void setLastName(String last)
41 {
42 lastName = last; // deve validar
43 } // fim do método setLastName
44
45 // retorna o sobrenome
46 public String getLastName()
47 {
48 return lastName;
49 } // fim do método getLastName
50
51 // configura o número da previdência social
52 public void setSocialSecurityNumber(String ssn)
53 {
54 socialSecurityNumber = ssn; // deve validar
55 } // fim do método setSocialSecurityNumber
56
57 // retorna o número da previdência social
58 public String getSocialSecurityNumber()
59 {
60 return socialSecurityNumber;

```

**Figura G.6** A classe `BasePlusCommissionEmployee` representa um funcionário que recebe um salário-base mais uma comissão. (*continua*)

```
61 } // fim do método getSocialSecurityNumber
62
63 // configura o valor das vendas brutas
64 public void setGrossSales(double sales)
65 {
66 if (sales >= 0.0)
67 grossSales = sales;
68 else
69 throw new IllegalArgumentException(
70 "Gross sales must be >= 0.0");
71 } // fim do método setGrossSales
72
73 // retorna o valor das vendas brutas
74 public double getGrossSales()
75 {
76 return grossSales;
77 } // fim do método getGrossSales
78
79 // configura a taxa de comissão
80 public void setCommissionRate(double rate)
81 {
82 if (rate > 0.0 && rate < 1.0)
83 commissionRate = rate;
84 else
85 throw new IllegalArgumentException(
86 "Commission rate must be > 0.0 and < 1.0");
87 } // fim do método setCommissionRate
88
89 // retorna a taxa de comissão
90 public double getCommissionRate()
91 {
92 return commissionRate;
93 } // fim do método getCommissionRate
94
95 // configura o salário-base
96 public void setBaseSalary(double salary)
97 {
98 if (salary >= 0.0)
99 baseSalary = salary;
100 else
101 throw new IllegalArgumentException(
102 "Base salary must be >= 0.0");
103 } // fim do método setBaseSalary
104
105 // retorna o salário-base
106 public double getBaseSalary()
107 {
108 return baseSalary;
109 } // fim do método getBaseSalary
110
111 // calcula os ganhos
112 public double earnings()
113 {
114 return baseSalary + (commissionRate * grossSales);
115 } // fim do método earnings
116
117 // retorna a representação de String de BasePlusCommissionEmployee
118 @Override // indica que esse método sobrescreve um método da superclasse
119 public String toString()
120 {
```

---

**Figura G.6** A classe `BasePlusCommissionEmployee` representa um funcionário que recebe um salário-base mais uma comissão. (*continua*)

```
121 return String.format(
122 "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f\n%s: %.2f",
123 "base-salaried commission employee", firstName, lastName,
124 "social security number", socialSecurityNumber,
125 "gross sales", grossSales, "commission rate", commissionRate,
126 "base salary", baseSalary);
127 } // fim do método toString
128 } // fim da classe BasePlusCommissionEmployee
```

**Figura G.6** A classe `BasePlusCommissionEmployee` representa um funcionário que recebe um salário-base mais uma comissão.

A classe `BasePlusCommissionEmployee` não especifica “`extends Object`” na linha 5; portanto, ela estende `Object` implicitamente. Além disso, como o construtor da classe `CommissionEmployee` (linhas 13 a 22 da Fig. G.4), o construtor da classe `BasePlusCommissionEmployee` chama o construtor padrão da classe `Object` implicitamente, conforme registrado no comentário da linha 18.

O método `earnings` da classe `BasePlusCommissionEmployee` (linhas 112 a 115) retorna o resultado da soma do salário-base de `BasePlusCommissionEmployee` ao produto da taxa de comissão vezes as vendas brutas do funcionário.

A classe `BasePlusCommissionEmployee` sobrescreve o método `toString` de `Object` para retornar uma `String` contendo as informações de `BasePlusCommissionEmployee`. Mais uma vez, usamos o especificador de formato `%.2f` para formatar as vendas brutas, a taxa de comissão e o salário-base com dois dígitos de precisão à direita do ponto decimal (linha 122).

### Testando a classe `BasePlusCommissionEmployee`

A Figura G.7 testa a classe `BasePlusCommissionEmployee`. As linhas 9 a 11 criam um objeto `BasePlusCommissionEmployee` e passam “Bob”, “Lewis”, “333-33-3333”, 5000, .04 e 300 para o construtor como nome, sobrenome, número da previdência social, vendas brutas, taxa de comissão e salário-base, respectivamente. As linhas 16 a 27 usam os métodos `get` de `BasePlusCommissionEmployee` para recuperar os valores das variáveis de instância do objeto a fim de gerar a saída. A linha 29 chama o método `setBaseSalary` do objeto para alterar o salário-base. O método `setBaseSalary` (Fig. G.6, linhas 96 a 103) garante que a variável de instância `baseSalary` não receba um valor negativo. As linhas 31 a 33 da Fig. G.7 chamam o método `toString` explicitamente para obter a representação de `String` do objeto.

### Observações sobre a classe `BasePlusCommissionEmployee`

Grande parte do código da classe `BasePlusCommissionEmployee` (Fig. G.6) é semelhante ou idêntica ao da classe `CommissionEmployee` (Fig. G.4). Por exemplo, as variáveis de instância `private firstName` e `lastName` e os métodos `setFirstName`, `getFirstName`, `setLastName` e `getLastName` são iguais aos da classe `CommissionEmployee`. As duas classes também contêm as variáveis de instância `private socialSecurityNumber`, `commissionRate` e `grossSales`, e métodos `get` e `set` correspondentes. Além disso, o construtor de `BasePlusCommissionEmployee` é quase idêntico ao da classe `CommissionEmployee`, exceto que o construtor de `BasePlusCommissionEmployee` também configura `baseSalary`. As outras adições feitas à classe `BasePlusCommissionEmployee` são a variável de instância `private baseSalary` e os métodos `setBaseSalary` e `getBaseSalary`. O método `toString` da classe `BasePlusCommissionEmployee` é praticamente idêntico ao da classe `CommissionEmployee`, exceto que também gera a variável de instância `baseSalary` com dois dígitos de precisão à direita do ponto decimal.

```

1 // Fig. G.7: BasePlusCommissionEmployeeTest.java
2 // Programa de teste de BasePlusCommissionEmployee.
3
4 public class BasePlusCommissionEmployeeTest
5 {
6 public static void main(String[] args)
7 {
8 // instancia objeto BasePlusCommissionEmployee
9 BasePlusCommissionEmployee employee =
10 new BasePlusCommissionEmployee(
11 "Bob", "Lewis", "333-33-3333", 5000, .04, 300);
12
13 // obtém dados de funcionário assalariado
14 System.out.println(
15 "Employee information obtained by get methods: \n");
16 System.out.printf("%s %s\n", "First name is",
17 employee.getFirstName());
18 System.out.printf("%s %s\n", "Last name is",
19 employee.getLastName());
20 System.out.printf("%s %s\n", "Social security number is",
21 employee.getSocialSecurityNumber());
22 System.out.printf("%s %.2f\n", "Gross sales is",
23 employee.getGrossSales());
24 System.out.printf("%s %.2f\n", "Commission rate is",
25 employee.getCommissionRate());
26 System.out.printf("%s %.2f\n", "Base salary is",
27 employee.getBaseSalary());
28
29 employee.setBaseSalary(1000); // configura o salário-base
30
31 System.out.printf("\n%s:\n\n%s\n",
32 "Updated employee information obtained by toString",
33 employee.toString());
34 } // fim de main
35 } // fim da classe BasePlusCommissionEmployeeTest

```

```

Employee information obtained by get methods:
First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Updated employee information obtained by toString:
base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00

```

**Figura G.7** Programa de teste de BasePlusCommissionEmployee.

Literalmente, *copiamos* o código da classe `CommissionEmployee` e o *colamos* na classe `BasePlusCommissionEmployee`; então, modificamos a classe `BasePlusCommissionEmployee` para incluir um salário-base e métodos que o manipulam. Essa *estratégia de “copiar e colar”* muitas vezes é propensa a erros e demorada. Pior ainda, ela espalha cópias do mesmo código por todo o sistema, criando um pesadelo de manutenção de código. Será que há um modo de “absorver” as variáveis de instância e os métodos de uma classe, de maneira

que eles façam parte de outras classes, *sem duplicar código*? A seguir, respondemos a essa pergunta usando uma estratégia mais elegante para construir classes, a qual enfatiza as vantagens da herança.



#### Observação sobre engenharia de software G.1

*Com a herança, as variáveis de instância e os métodos comuns de todas as classes da hierarquia são declarados em uma superclasse. Quando são feitas alterações nesses recursos comuns na superclasse, as subclasses as herdam. Sem a herança, as alterações precisariam ser feitas em todos os arquivos de código-fonte que contivessem uma cópia do código em questão.*

### G.4.3 Criação de uma hierarquia de herança CommissionEmployee–BasePlusCommissionEmployee

Agora, redeclaramos a classe BasePlusCommissionEmployee (Fig. G.8) para *estender* a classe CommissionEmployee (Fig. G.4). Um objeto BasePlusCommissionEmployee *é um* CommissionEmployee, pois a herança transmite os recursos da classe CommissionEmployee. A classe BasePlusCommissionEmployee também tem a variável de instância baseSalary (Fig. G.8, linha 6).

A palavra-chave extends (linha 4) indica herança. BasePlusCommissionEmployee *herda* as variáveis de instância e os métodos de CommissionEmployee, mas somente os membros public e protected da superclasse são diretamente acessíveis na subclasse. O construtor de CommissionEmployee *não* é herdado. Assim, os serviços public de BasePlusCommissionEmployee incluem seu construtor (linhas 9 a 16), os métodos public herdados de CommissionEmployee e os métodos setBaseSalary (linhas 19 a 26), getBaseSalary (linhas 29 a 32), earnings (linhas 35 a 40) e toString (linhas 43 a 53). Os métodos earnings e toString *sobrescrevem* os métodos correspondentes na classe CommissionEmployee, pois suas versões de superclasse não calculam os ganhos de um BasePlusCommissionEmployee corretamente nem retornam uma representação de String apropriada.

```

1 // Fig. G.8: BasePlusCommissionEmployee.java
2 // Os membros private da superclasse não podem ser acessados em uma subclasse.
3
4 public class BasePlusCommissionEmployee extends CommissionEmployee
5 {
6 private double baseSalary; // salário-base por semana
7
8 // construtor de seis argumentos
9 public BasePlusCommissionEmployee(String first, String last,
10 String ssn, double sales, double rate, double salary)
11 {
12 // chamada explícita para o construtor CommissionEmployee da superclasse
13 super(first, last, ssn, sales, rate);
14
15 setBaseSalary(salary); // valida e armazena o salário-base
16 } // fim do construtor de seis argumentos de BasePlusCommissionEmployee
17
18 // configura o salário-base
19 public void setBaseSalary(double salary)
20 {
21 if (salary >= 0.0)
22 baseSalary = salary;
23 else
24 throw new IllegalArgumentException(

```

**Figura G.8** Os membros private da superclasse não podem ser acessados em uma subclasse. (continua)



```

25 "Base salary must be >= 0.0");
26 } // fim do método setBaseSalary
27
28 // retorna o salário-base
29 public double getBaseSalary()
30 {
31 return baseSalary;
32 } // fim do método getBaseSalary
33
34 // calcula os ganhos
35 @Override // indica que esse método sobrescreve um método da superclasse
36 public double earnings()
37 {
38 // não permitido: commissionRate e grossSales são private na superclasse
39 return baseSalary + (commissionRate * grossSales);
40 } // fim do método earnings
41
42 // retorna a representação de String de BasePlusCommissionEmployee
43 @Override // indica que esse método sobrescreve um método da superclasse
44 public String toString()
45 {
46 // não permitido: tentativas de acessar membros private da superclasse
47 return String.format(
48 "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f\n%s: %.2f",
49 "base-salaried commission employee", firstName, lastName,
50 "social security number", socialSecurityNumber,
51 "gross sales", grossSales, "commission rate", commissionRate,
52 "base salary", baseSalary);
53 } // fim do método toString
54 } // fim da classe BasePlusCommissionEmployee

```

```

BasePlusCommissionEmployee.java:39: commissionRate has private access in
CommissionEmployee
 return baseSalary + (commissionRate * grossSales);
 ^
BasePlusCommissionEmployee.java:39: grossSales has private access in
CommissionEmployee
 return baseSalary + (commissionRate * grossSales);
 ^
BasePlusCommissionEmployee.java:49: firstName has private access in
CommissionEmployee
 "base-salaried commission employee", firstName, lastName,
 ^
BasePlusCommissionEmployee.java:49: lastName has private access in
CommissionEmployee
 "base-salaried commission employee", firstName, lastName,
 ^
BasePlusCommissionEmployee.java:50: socialSecurityNumber has private access
in CommissionEmployee
 "social security number", socialSecurityNumber,
 ^
BasePlusCommissionEmployee.java:51: grossSales has private access in
CommissionEmployee
 "gross sales", grossSales, "commission rate", commissionRate,
 ^
BasePlusCommissionEmployee.java:51: commissionRate has private access in
CommissionEmployee
 "gross sales", grossSales, "commission rate", commissionRate,
 ^
7 errors

```

**Figura G.8** Os membros private da superclasse não podem ser acessados em uma subclasse.

***O construtor de uma subclasse deve chamar o construtor de sua superclasse***

O construtor de cada subclasse deve chamar, implícita ou explicitamente, o construtor de sua superclasse para inicializar as variáveis de instância herdadas da superclasse. A linha 13 no construtor de seis argumentos de `BasePlusCommissionEmployee` (linhas 9 a 16) chama explicitamente o construtor de cinco argumentos da classe `CommissionEmployee` (declarada nas linhas 13 a 22 da Fig. G.4) para inicializar a parte da superclasse de um objeto `BasePlusCommissionEmployee` (isto é, as variáveis `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` e `commissionRate`). Fazemos isso usando a **sintaxe de chamada de construtor de superclasse** – a palavra-chave `super`, seguida por um conjunto de parênteses contendo os argumentos do construtor da superclasse. Os argumentos `first`, `last`, `ssn`, `sales` e `rate` são usados para inicializar os membros da superclasse `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` e `commissionRate`, respectivamente. Se o construtor de `BasePlusCommissionEmployee` não chamasse o construtor da superclasse explicitamente, a linguagem Java tentaria chamar o construtor sem argumentos ou o construtor padrão da superclasse. A classe `CommissionEmployee` não tem tal construtor, de modo que o compilador emitiria um erro. A chamada explícita do construtor da superclasse na linha 13 da Fig. G.8 deve ser a *primeira* instrução no corpo do construtor da subclasse. Quando uma superclasse contém um construtor sem argumentos, você pode usar `super()` para chamá-lo explicitamente, mas isso raramente é feito.

***Método `Earnings` de `BasePlusCommissionEmployee`***

O compilador gera erros para a linha 39 porque as variáveis de instância `commissionRate` e `grossSales` da superclasse `CommissionEmployee` são `private` – os métodos da subclasse `BasePlusCommissionEmployee` não podem acessar variáveis de instância `private` da superclasse `CommissionEmployee`. Realçamos o código errado. O compilador emite mais erros nas linhas 49 a 51 do método `toString` de `BasePlusCommissionEmployee` pelo mesmo motivo. Os erros em `BasePlusCommissionEmployee` poderiam ser evitados pelo uso dos métodos `get` herdados da classe `CommissionEmployee`. Por exemplo, a linha 39 poderia ter usado `getCommissionRate` e `getGrossSales` para acessar as variáveis de instância `private` `commissionRate` e `grossSales` de `CommissionEmployee`, respectivamente. As linhas 49 a 51 também poderiam ter usado métodos `get` apropriados para recuperar os valores das variáveis de instância da superclasse.

**G.4.4 Hierarquia de herança****CommissionEmployee–BasePlusCommissionEmployee usando variáveis de instância `protected`**

Para permitir que a classe `BasePlusCommissionEmployee` acesse diretamente as variáveis de instância `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` e `commissionRate` da superclasse, podemos declarar esses membros como `protected` na superclasse. Conforme discutimos na seção G.3, os membros `protected` de uma superclasse são acessíveis para todas as subclasses dessa superclasse. Na nova classe `CommissionEmployee`, modificamos somente as linhas 6 a 10 da Fig. G.4 para declararmos as variáveis de instância com o modificador de acesso `protected`, como segue:

```
protected String firstName;
protected String lastName;
protected String socialSecurityNumber;
protected double grossSales; // vendas brutas semanais
protected double commissionRate; // porcentagem de comissão
```

O restante da declaração de classe (que não aparece aqui) é idêntico ao da Fig. G.4.

Poderíamos ter declarado as variáveis de instância de `CommissionEmployee` como `public` para permitir à subclasse `BasePlusCommissionEmployee` acessá-las. No entanto, declarar variáveis de instância `public` é engenharia de software ruim, pois possibilita acesso irrestrito a essas variáveis, aumentando substancialmente as chances de erros. Com variáveis de instância `protected`, a subclasse obtém acesso às variáveis de instância, mas as classes que não são subclasses e as que não estão no mesmo pacote não podem acessá-las diretamente – lembre-se de que os membros de classe `protected` também são visíveis para outras classes do mesmo pacote.

### Classe *BasePlusCommissionEmployee*

A classe `BasePlusCommissionEmployee` (Fig. G.9) estende a nova versão da classe `CommissionEmployee` com variáveis de instância `protected`. Os objetos `BasePlusCommissionEmployee` herdam as variáveis de instância `protected` `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` e `commissionRate` de `CommissionEmployee` – todas essas variáveis agora são membros `protected` de `BasePlusCommissionEmployee`. Como resultado, o compilador não gera erros ao compilar a linha 37 do método `earnings` e as linhas 46 a 48 do método `toString`. Se outra classe estender essa versão da classe `BasePlusCommissionEmployee`, a nova subclasse também poderá acessar os membros `protected`.

```

1 // Fig. G.9: BasePlusCommissionEmployee.java
2 // BasePlusCommissionEmployee herda variáveis de
3 // instância protected de CommissionEmployee.
4
5 public class BasePlusCommissionEmployee extends CommissionEmployee
6 {
7 private double baseSalary; // salário-base por semana
8
9 // construtor de seis argumentos
10 public BasePlusCommissionEmployee(String first, String last,
11 String ssn, double sales, double rate, double salary)
12 {
13 super(first, last, ssn, sales, rate);
14 setBaseSalary(salary); // valida e armazena o salário-base
15 } // fim do construtor de seis argumentos de BasePlusCommissionEmployee
16
17 // configura o salário-base
18 public void setBaseSalary(double salary)
19 {
20 if (salary >= 0.0)
21 baseSalary = salary;
22 else
23 throw new IllegalArgumentException(
24 "Base salary must be >= 0.0");
25 } // fim do método setBaseSalary
26
27 // retorna o salário-base
28 public double getBaseSalary()
29 {
30 return baseSalary;
31 } // fim do método getBaseSalary
32
33 // calcula os ganhos
34 @Override // indica que esse método sobreescreve um método da superclasse

```

**Figura G.9** `BasePlusCommissionEmployee` herda variáveis de instância `protected` de `CommissionEmployee`.

```
35 public double earnings()
36 {
37 return baseSalary + (commissionRate * grossSales);
38 } // fim do método earnings
39
40 // retorna a representação de String de BasePlusCommissionEmployee
41 @Override // indica que esse método sobrescreve um método da superclasse
42 public String toString()
43 {
44 return String.format(
45 "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f\n%s: %.2f",
46 "base-salaried commission employee", firstName, lastName,
47 "social security number", socialSecurityNumber,
48 "gross sales", grossSales, "commission rate", commissionRate,
49 "base salary", baseSalary);
50 } // fim do método toString
51 } // fim da classe BasePlusCommissionEmployee
```

**Figura G.9** BasePlusCommissionEmployee herda variáveis de instância protected de CommissionEmployee.

Quando você cria um objeto BasePlusCommissionEmployee, ele contém todas as variáveis de instância declaradas na hierarquia de classes até esse ponto – ou seja, as variáveis de instância das classes Object, CommissionEmployee e BasePlusCommissionEmployee. A classe BasePlusCommissionEmployee não herda o construtor da classe CommissionEmployee. Contudo, o construtor de seis argumentos da classe BasePlusCommissionEmployee (linhas 10 a 15) chama o construtor de cinco argumentos da classe CommissionEmployee *explicitamente* para inicializar as variáveis de instância herdadas da classe CommissionEmployee por BasePlusCommissionEmployee. Do mesmo modo, o construtor da classe CommissionEmployee chama o construtor da classe Object *implicitamente*. O construtor de BasePlusCommissionEmployee precisa fazer isso *explicitamente* porque CommissionEmployee *não* fornece um construtor sem argumentos que possa ser chamado implicitamente.

### Testando a classe BasePlusCommissionEmployee

A classe BasePlusCommissionEmployeeTest deste exemplo é idêntica à da Fig. G.7 e produz a mesma saída; portanto, não a mostramos aqui. Embora a versão da classe BasePlusCommissionEmployee da Fig. G.6 não utilize herança e a versão da Fig. G.9 utilize, *as duas classes fornecem a mesma funcionalidade*. O código-fonte da Fig. G.9 (51 linhas) é consideravelmente menor que o da Fig. G.6 (128 linhas), pois a maior parte da funcionalidade de BasePlusCommissionEmployee agora é herdada de CommissionEmployee – agora existe apenas uma cópia da funcionalidade de CommissionEmployee. Isso torna o código mais fácil de ser mantido, modificado e depurado, pois o código relacionado a um funcionário comissionado existe apenas na classe CommissionEmployee.

### Observações sobre o uso de variáveis de instância protected

Neste exemplo, declaramos as variáveis de instância da superclasse como protected para que as subclasses pudessem acessá-las. Herdar variáveis de instância protected aumenta um pouco o desempenho, pois podemos acessá-las diretamente na subclasse sem incorrer na sobrecarga de uma chamada de método *set* ou *get*. No entanto, na maioria dos casos é melhor usar variáveis de instância private para estimular a engenharia de software correta e deixar os problemas de otimização de código para o compilador. Seu código será mais fácil de ser mantido, modificado e depurado. O uso de variáveis de instância

`protected` gera vários problemas em potencial. Primeiramente, o objeto da subclasse pode configurar o valor de uma variável herdada diretamente, sem usar um método `set`. Portanto, um objeto da subclasse pode atribuir um valor inválido à variável, possivelmente deixando o objeto em um estado incoerente. Por exemplo, se declarássemos a variável de instância `grossSales` de `CommissionEmployee` como `protected`, um objeto da subclasse (por exemplo, `BasePlusCommissionEmployee`) poderia atribuir um valor negativo a `grossSales`. Outro problema no uso de variáveis de instância `protected` é que os métodos da subclasse provavelmente serão escritos de modo a depender da implementação dos dados da superclasse. Na prática, as subclasses devem depender somente dos serviços da superclasse (isto é, métodos não `private`) e não de sua implementação de dados. Com variáveis de instância `protected` na superclasse, talvez seja necessário modificar todas as subclasses da superclasse, caso a implementação desta mude. Por exemplo, se mudássemos os nomes das variáveis de instância `firstName` e `lastName` para `first` e `last` por algum motivo, então teríamos de fazer isso para todas as ocorrências nas quais uma subclasse referenciasse diretamente as variáveis de instância `firstName` e `lastName` da superclasse. Nesse caso, diz-se que o software é **frágil** ou **delicado**, pois uma pequena alteração na superclasse pode “quebrar” a implementação da subclasse. É necessário fazer que uma alteração na implementação da superclasse ainda forneça os mesmos serviços para as subclasses. Evidentemente, se os serviços da superclasse mudarem, precisaremos reimplementar nossas subclasses. Um terceiro problema é que os membros `protected` de uma classe são visíveis para todas as classes do mesmo pacote da classe que os contém – isso nem sempre é desejável.



#### Observação sobre engenharia de software G.2

*Use o modificador de acesso `protected` quando uma superclasse precisar fornecer um método apenas para suas subclasses e para outras classes do mesmo pacote, mas não para outros clientes.*



#### Observação sobre engenharia de software G.3

*Declarar as variáveis de instância da superclasse como `private` (em oposição a `protected`) permite que a implementação da superclasse dessas variáveis de instância mude sem afetar as implementações de subclasse.*



#### Dica de prevenção de erro G.1

*Quando possível, não inclua variáveis de instância `protected` em uma superclasse. Em vez disso, inclua métodos não `private` que acessem variáveis de instância `private`. Isso ajudará a garantir que os objetos da classe mantenham estados coerentes.*

### G.4.5 Hierarquia de herança `CommissionEmployee`–`BasePlusCommissionEmployee` usando variáveis de instância `private`

Vamos examinar novamente nossa hierarquia, desta vez usando boas práticas de engenharia de software. A classe `CommissionEmployee` (Fig. G.10) declara as variáveis de instância `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` e `commissionRate` como `private` (linhas 6 a 10) e fornece os métodos `public` `setFirstName`, `getFirstName`, `setLastName`, `getLastName`, `setSocialSecurityNumber`, `getSocialSecurityNumber`, `setGrossSales`, `getGrossSales`, `setCommissionRate`, `getCommissionRate`, `earnin-`

gs e toString para manipular esses valores. Os métodos earnings (linhas 93 a 96) e toString (linhas 99 a 107) usam os métodos *get* da classe para obter os valores de suas variáveis de instância. Se decidirmos mudar os nomes das variáveis de instância, as declarações de earnings e toString não exigirão modificação – somente os corpos dos métodos *get* e *set* que manipulam as variáveis de instância diretamente precisarão mudar. Essas alterações ocorrem unicamente dentro da superclasse – nenhuma alteração é necessária na subclasse. *Delimitar os efeitos de alterações* como essas é uma boa prática de engenharia de software.

```

1 // Fig. G.10: CommissionEmployee.java
2 // A classe CommissionEmployee usa métodos para manipular suas
3 // variáveis de instância private.
4 public class CommissionEmployee
5 {
6 private String firstName;
7 private String lastName;
8 private String socialSecurityNumber;
9 private double grossSales; // vendas brutas semanais
10 private double commissionRate; // porcentagem de comissão
11
12 // construtor de cinco argumentos
13 public CommissionEmployee(String first, String last, String ssn,
14 double sales, double rate)
15 {
16 // a chamada implícita ao construtor de Object ocorre aqui
17 firstName = first;
18 lastName = last;
19 socialSecurityNumber = ssn;
20 setGrossSales(sales); // valida e armazena as vendas brutas
21 setCommissionRate(rate); // valida e armazena a taxa de comissão
22 } // fim do construtor de cinco argumentos de CommissionEmployee
23
24 // configura o nome
25 public void setFirstName(String first)
26 {
27 firstName = first; // deve validar
28 } // fim do método setFirstName
29
30 // retorna o nome
31 public String getFirstName()
32 {
33 return firstName;
34 } // fim do método getFirstName
35
36 // configura o sobrenome
37 public void setLastName(String last)
38 {
39 la5stName = last; // deve validar
40 } // fim do método setLastName
41
42 // retorna o sobrenome
43 public String getLastName()
44 {
45 return lastName;
46 } // fim do método getLastName
47
48 // configura o número da previdência social

```

**Figura G.10** A classe CommissionEmployee usa métodos para manipular suas variáveis de instância private. (continua)

```
49 public void setSocialSecurityNumber(String ssn)
50 {
51 socialSecurityNumber = ssn; // deve validar
52 } // fim do método setSocialSecurityNumber
53
54 // retorna o número da previdência social
55 public String getSocialSecurityNumber()
56 {
57 return socialSecurityNumber;
58 } // fim do método getSocialSecurityNumber
59
60 // configura o valor das vendas brutas
61 public void setGrossSales(double sales)
62 {
63 if (sales >= 0.0)
64 grossSales = sales;
65 else
66 throw new IllegalArgumentException(
67 "Gross sales must be >= 0.0");
68 } // fim do método setGrossSales
69
70 // retorna o valor das vendas brutas
71 public double getGrossSales()
72 {
73 return grossSales;
74 } // fim do método getGrossSales
75
76 // configura a taxa de comissão
77 public void setCommissionRate(double rate)
78 {
79 if (rate > 0.0 && rate < 1.0)
80 commissionRate = rate;
81 else
82 throw new IllegalArgumentException(
83 "Commission rate must be > 0.0 and < 1.0");
84 } // fim do método setCommissionRate
85
86 // retorna a taxa de comissão
87 public double getCommissionRate()
88 {
89 return commissionRate;
90 } // fim do método getCommissionRate
91
92 // calcula os ganhos
93 public double earnings()
94 {
95 return getCommissionRate() * getGrossSales();
96 } // fim do método earnings
97
98 // retorna representação de String do objeto CommissionEmployee
99 @Override // indica que esse método sobreescreve um método da superclasse
100 public String toString()
101 {
102 return String.format("%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f",
103 "commission employee", getFirstName(), getLastName(),
104 "social security number", getSocialSecurityNumber(),
105 "gross sales", getGrossSales(),
106 "commission rate", getCommissionRate());
107 } // fim do método toString
108 } // fim da classe CommissionEmployee
```

---

**Figura G.10** A classe `CommissionEmployee` usa métodos para manipular suas variáveis de instância `private`.

A subclasse `BasePlusCommissionEmployee` (Fig. G.11) herda os métodos não `private` de `CommissionEmployee` e pode acessar os membros `private` da superclasse por meio desses métodos. A classe `BasePlusCommissionEmployee` tem várias mudanças que a tornam diferente da Fig. G.9. Os métodos `earnings` (linhas 35 a 39) e `toString` (linhas 42 a 47) chamam o método `getBaseSalary` para obter o valor do salário-base, em vez de acessarem `baseSalary` diretamente. Se decidirmos mudar o nome da variável de instância `baseSalary`, somente os corpos dos métodos `setBaseSalary` e `getBaseSalary` precisarão mudar.

```
1 // Fig. G.11: BasePlusCommissionEmployee.java
2 // A classe BasePlusCommissionEmployee herda de CommissionEmployee
3 // e acessa os dados private da superclasse por meio de
4 // métodos public herdados.
5
6 public class BasePlusCommissionEmployee extends CommissionEmployee
7 {
8 private double baseSalary; // salário-base por semana
9
10 // construtor de seis argumentos
11 public BasePlusCommissionEmployee(String first, String last,
12 String ssn, double sales, double rate, double salary)
13 {
14 super(first, last, ssn, sales, rate);
15 setBaseSalary(salary); // valida e armazena o salário-base
16 } // fim do construtor de seis argumentos de BasePlusCommissionEmployee
17
18 // configura o salário-base
19 public void setBaseSalary(double salary)
20 {
21 if (salary >= 0.0)
22 baseSalary = salary;
23 else
24 throw new IllegalArgumentException(
25 "Base salary must be >= 0.0");
26 } // fim do método setBaseSalary
27
28 // retorna o salário-base
29 public double getBaseSalary()
30 {
31 return baseSalary;
32 } // fim do método getBaseSalary
33
34 // calcula os ganhos
35 @Override // indica que esse método sobreescreve um método da superclasse
36 public double earnings()
37 {
38 return getBaseSalary() + super.earnings();
39 } // fim do método earnings
40
41 // retorna a representação de String de BasePlusCommissionEmployee
42 @Override // indica que esse método sobreescreve um método da superclasse
43 public String toString()
44 {
45 return String.format("%s %s\n%s: %.2f", "base-salaried",
46 super.toString(), "base salary", getBaseSalary());
47 } // fim do método toString
48 } // fim da classe BasePlusCommissionEmployee
```

**Figura G.11** A classe `BasePlusCommissionEmployee` herda de `CommissionEmployee` e acessa os dados `private` da superclasse por meio de métodos `public` herdados.



### **Método earnings da classe BasePlusCommissionEmployee**

O método `earnings` (linhas 35 a 39) sobrescreve o método `earnings` da classe `CommissionEmployee` (Fig. G.10, linhas 93 a 96) para calcular os ganhos de um funcionário assalariado. A nova versão obtém a parte dos ganhos com base apenas na comissão, chamando o método `earnings` de `CommissionEmployee` com `super.earnings()` (linha 38); então, soma o salário-base a esse valor para calcular os ganhos totais. Observe a sintaxe usada para chamar um método de superclasse sobrescrito a partir de uma subclasse – coloca-se a palavra-chave `super` e um separador ponto (.) antes do nome do método da superclasse. Essa chamada de método é uma boa prática de engenharia de software – se um método executa todas ou parte das ações necessárias para outro método, chame esse método, em vez de duplicar seu código. Fazendo que o método `earnings` de `BasePlusCommissionEmployee` chame o método `earnings` de `CommissionEmployee` para calcular parte dos ganhos de um objeto `BasePlusCommissionEmployee`, evitamos duplicação de código e reduzimos os problemas de manutenção de código. Se não usássemos "super.", o método `earnings` de `BasePlusCommissionEmployee` chamaria a si mesmo, em vez da versão da superclasse. Isso resultaria em um fenômeno chamado *recursão infinita*, o qual finalmente faria a pilha de chamadas de método estourar – um erro de tempo de execução (Runtime) fatal.

### **Método toString da classe BasePlusCommissionEmployee**

Do mesmo modo, o método `toString` de `BasePlusCommissionEmployee` (Fig. G.11, linhas 42 a 47) sobrescreve o método `toString` da classe `CommissionEmployee` (Fig. G.10, linhas 99 a 107) para retornar uma representação de `String` adequada para um funcionário assalariado. A nova versão cria parte da representação de `String` de um objeto `BasePlusCommissionEmployee` (isto é, a `String` "commission employee" e os valores das variáveis de instância private da classe `CommissionEmployee`) chamando o método `toString` de `CommissionEmployee` com a expressão `super.toString()` (Fig. G.11, linha 46). Então, o método `toString` de `BasePlusCommissionEmployee` gera o restante da representação de `String` de um objeto `BasePlusCommissionEmployee` (isto é, o valor do salário-base da classe `BasePlusCommissionEmployee`).



#### **Erro de programação comum G.2**

*Quando um método de superclasse é sobrescrito em uma subclasse, a versão da subclasse frequentemente chama a versão da superclasse para fazer parte do trabalho. Deixar de prefixar o nome do método da superclasse com a palavra-chave `super` e um separador ponto (.) ao chamar o método da superclasse faz que o método da subclasse chame a si mesmo, possivelmente gerando um erro chamado *recursão infinita*. Usada corretamente, a recursão é um recurso poderoso.*

### **Testando a classe BasePlusCommissionEmployee**

A classe `BasePlusCommissionEmployeeTest` realiza em um objeto `BasePlusCommissionEmployee` as mesmas manipulações da Fig. G.7 e produz a mesma saída; portanto, não a mostramos aqui. Embora cada classe `BasePlusCommissionEmployee` examinada tenha comportamento idêntico, a versão da Fig. G.11 tem um projeto melhor. Usando herança e chamando métodos que ocultam os dados e garantem a coerência, construímos, de forma eficiente e eficaz, uma classe bem projetada.

## G.5 Classe Object

Conforme discutimos anteriormente neste apêndice, todas as classes em Java herdam direta ou indiretamente da classe `Object` (pacote `java.lang`); portanto, seus 11 métodos (alguns sobrecarregados) são herdados por todas as outras classes. A Figura G.12 resume os métodos de `Object`. Discutimos vários métodos de `Object` por todo o livro (conforme indicado na Fig. G.12).

| Método                                                           | Descrição                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>clone</code>                                               | Este método <code>protected</code> , que não recebe argumentos e retorna uma referência de <code>Object</code> , faz uma cópia do objeto em que é chamado. A implementação padrão faz a assim chamada <b>cópia rasa</b> – os valores de variável de instância de um objeto são copiados em outro objeto do mesmo tipo. Para tipos de referência, somente as referências são copiadas. Uma implementação típica do método <code>clone</code> sobrescrito faria uma <b>cópia profunda</b> , a qual cria um novo objeto para cada variável de instância de tipo de referência. É difícil implementar <code>clone</code> corretamente. Por isso, seu uso é desestimulado. Muitos especialistas da área sugerem usar serialização de objetos em seu lugar. Apresentamos a serialização de objetos no Apêndice J. |
| <code>equals</code>                                              | Este método compara dois objetos quanto à igualdade e retorna <code>true</code> se são iguais; caso contrário, retorna <code>false</code> . O método recebe qualquer <code>Object</code> como argumento. Quando objetos de uma classe em particular precisam ser comparados quanto à igualdade, a classe deve sobrescrever o método <code>equals</code> para comparar o <i>conteúdo</i> dos dois objetos. A implementação padrão de <code>equals</code> usa o operador <code>==</code> para determinar se duas referências se referem ao mesmo objeto na memória.                                                                                                                                                                                                                                           |
| <code>finalize</code>                                            | Este método <code>protected</code> é chamado pelo coletor de lixo (apresentado na seção F.9) para fazer a limpeza final em um objeto, imediatamente antes de o coletor recuperar a memória do objeto. Lembre-se de que não está claro se ou quando o método <code>finalize</code> será chamado. Por isso, a maioria dos programadores deve evitar esse método.                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <code>getClass</code>                                            | Em Java, todo objeto sabe seu próprio tipo no momento da execução. O método <code>getClass</code> retorna um objeto da classe <code>Class</code> (pacote <code>java.lang</code> ) que contém informações sobre o tipo do objeto, como seu nome de classe (retornado pelo método <code>getName</code> de <code>Class</code> ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <code>hashCode</code>                                            | Códigos hash (hashcodes) são valores <code>int</code> úteis para acelerar o armazenamento e a recuperação de informações armazenadas em uma estrutura de dados conhecida como tabela hash (discutida na seção J.9). Esse método também é chamado como parte da implementação do método <code>toString</code> padrão da classe <code>Object</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>wait</code> , <code>notify</code> , <code>notifyAll</code> | Os métodos <code>notify</code> , <code>notifyAll</code> e as três versões sobrecarregadas de <code>wait</code> estão relacionados ao <code>multithread</code> , o qual é discutido no Apêndice J.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <code>toString</code>                                            | Este método (apresentado na seção G.4.1) retorna uma representação de <code>String</code> de um objeto. A implementação padrão retorna o nome do pacote e o nome da classe do objeto, seguidos de uma representação em hexadecimal do valor retornado pelo método <code>hashCode</code> do objeto.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

**Figura G.12** Métodos de `Object`.

Lembre, do Apêndice E, que arrays são objetos. Como resultado, assim como todos os outros objetos, os arrays herdam os membros da classe `Object`. Todo array tem um método `clone` sobrescrito que copia o array. Contudo, se o array armazena referências para objetos, os objetos não são copiados – é feita uma *cópia rasa*.

## G.6 Introdução ao polimorfismo

Continuamos nosso estudo sobre programação orientada a objetos explicando e demonstrando o **polimorfismo** com hierarquias de herança. O polimorfismo permite que você “programe no geral”, em vez de “programar no específico”. Em particular, o polimorfismo permite escrever programas que processam objetos que compartilham a mesma superclasse (direta ou indiretamente), como se todos fossem objetos da superclasse; isso pode simplificar a programação.

Considere o exemplo de polimorfismo a seguir. Suponha que criemos um programa para simular o movimento de vários tipos de animais para um estudo biológico. As classes `Peixe`, `Rã` e `Pássaro` representam os tipos de animais sob investigação. Imagine que cada classe estende a superclasse `Animal`, a qual contém um método `mover` e mantém a posição atual de um animal como coordenadas  $x$ - $y$ . Cada subclasse implementa o método `mover`. Nosso programa mantém um array `Animal` que contém referências para objetos de várias subclasses de `Animal`. Para simular os movimentos dos animais, o programa envia a cada objeto a *mesma* mensagem, uma vez por segundo – a saber, `mover`. Cada tipo específico de `Animal` responde a uma mensagem `mover` à sua própria maneira – um `Peixe` poderia nadar por um metro, uma `Rã` poderia pular por um metro e meio e um `Pássaro` poderia voar por três metros. Cada objeto sabe como modificar suas coordenadas  $x$ - $y$  apropriadamente para seu tipo *específico* de movimento. Contar com o fato de que cada objeto sabe como “fazer a coisa certa” (isto é, fazer o que é adequado para esse tipo de objeto) em resposta à mesma chamada de método é o principal conceito do polimorfismo. A mesma mensagem (neste caso, `mover`) enviada para uma variedade de objetos tem “muitas formas” de resultados – daí o termo polimorfismo.

### **Programar no específico**

Ocasionalmente, ao realizarmos processamento polimórfico, precisamos programar “no específico”. Vamos demonstrar que um programa pode determinar o tipo de um objeto *no momento da execução* e atuar nesse objeto de forma correspondente.

### **Interfaces**

O apêndice continua com uma introdução às interfaces Java. Uma interface descreve um conjunto de métodos que podem ser chamados em um objeto, mas *não* fornece implementações concretas para todos os métodos. Você pode declarar classes que **implementam** (isto é, fornecem implementações concretas para os métodos de) uma ou mais interfaces. Cada método de interface deve ser declarado em todas as classes que implementam a interface explicitamente. Quando uma classe implementa uma interface, todos os objetos dessa classe têm uma relação *é um* com o tipo de interface, e é garantido que todos os objetos da classe fornecem a funcionalidade descrita pela interface. Isso também vale para todas as subclasses dessa classe.

As interfaces são particularmente úteis para atribuir funcionalidade comum a classes possivelmente *não relacionadas*. Isso permite que objetos de classes não relacionadas sejam processados de modo polimórfico – objetos de classes que implementam a mesma interface podem responder a todas as chamadas de métodos da interface de forma personalizada. Para demonstrar a criação e o uso de interfaces, modificamos nosso aplicativo de folha de pagamento para criar um aplicativo de contas a pagar geral que pode calcular os pagamentos devidos aos funcionários da empresa e os valores das faturas a serem cobrados por bens adquiridos. Como você vai ver, as interfaces possibilitam recursos polimórficos semelhantes aos que são possíveis com a herança.

## G.7 Polimorfismo: um exemplo

### *Objetos espaciais em um videogame*

Suponha que projetemos um videogame que manipula objetos das classes `Marciano`, `Venusiano`, `Plutoniano`, `NaveEspacial` e `RaioLaser`. Imagine que cada classe herda da superclasse `ObjetoEspacial`, a qual contém o método `desenhar`. Cada subclasse implementa esse método. Um gerenciador de tela mantém uma coleção (por exemplo, um array `ObjetoEspacial`) de referências para objetos das várias classes. Para atualizar a tela, o gerenciador envia periodicamente a cada objeto a mesma mensagem – a saber, `desenhar`. Contudo, cada objeto responde à sua própria maneira, de acordo com sua classe. Por exemplo, um objeto `Marciano` poderia desenhar-se em vermelho com olhos verdes e com o número apropriado de antenas. Um objeto `NaveEspacial` poderia desenhar-se como um disco voador prata luminoso. Um objeto `RaioLaser` poderia desenhar-se como um feixe vermelho brilhante na tela. Novamente, a *mesma* mensagem (neste caso, `desenhar`) enviada para uma variedade de objetos tem “muitas formas” de resultados.

Um gerenciador de tela poderia usar polimorfismo para facilitar a inclusão de novas classes a um sistema, com modificações mínimas no código do sistema. Suponha que queiramos adicionar objetos `Mercuriano` ao nosso videogame. Para isso, construiríamos uma classe `Mercuriano` que estenderia `ObjetoEspacial` e forneceria sua própria implementação do método `desenhar`. Quando objetos `Mercuriano` aparecem na coleção `ObjetoEspacial`, o código do gerenciador de tela *chama o método desenhar, exatamente como faz para qualquer outro objeto da coleção, independentemente de seu tipo*. Assim, os novos objetos `Mercuriano` simplesmente “se conectam”, sem qualquer modificação no código do gerenciador de tela por parte do programador. Portanto, sem modificar o sistema (a não ser pela construção de novas classes e a modificação do código que cria novos objetos), você pode usar polimorfismo para convenientemente incluir tipos adicionais que não estavam previstos quando o sistema foi criado.



#### **Observação sobre engenharia de software G.4**

*O polimorfismo permite que você lide com as generalidades e deixe o ambiente de tempo de execução tratar dos detalhes específicos. Você pode fazer que os objetos se comportem de maneiras adequadas sem conhecer seus tipos (desde que os objetos pertençam à mesma hierarquia de herança).*



#### **Observação sobre engenharia de software G.5**

*O polimorfismo promove a extensibilidade: um software que ativa comportamento polimórfico é independente dos tipos de objeto para os quais as mensagens são enviadas. Novos tipos de objeto que podem responder às chamadas de método já existentes podem ser incorporados a um sistema sem modificar o sistema básico. Somente o código cliente que instancia novos objetos precisa ser modificado para acomodar os novos tipos.*

## G.8 Demonstração de comportamento polimórfico

A seção G.4 criou uma hierarquia de classes na qual a classe `BasePlusCommissionEmployee` herdava de `CommissionEmployee`. Os exemplos daquela seção manipulavam objetos `CommissionEmployee` e `BasePlusCommissionEmployee` usando referências a eles para cha-

mar seus métodos – apontamos variáveis de superclasse para objetos de superclasse e variáveis de subclasse para objetos de subclasse. Essas atribuições são naturais e simples – as variáveis de superclasse são *destinadas* a fazer referência a objetos de superclasse, e as variáveis de subclasse são *destinadas* a fazer referência a objetos de subclasse. Contudo, conforme você vai ver em breve, outras atribuições são possíveis.

No próximo exemplo, apontamos uma referência de *superclasse* para um objeto de *subclasse*. Então, mostramos como o fato de chamar um método em um objeto de subclasse por meio de uma referência de superclasse ativa a funcionalidade da *subclasse* – o tipo do *objeto referenciado*, não o tipo da *variável*, determina qual método é chamado. Esse exemplo demonstra que *um objeto de uma subclasse pode ser tratado como um objeto de sua superclasse*, permitindo várias manipulações interessantes. Um programa pode criar um array de variáveis de superclasse que se referem a objetos de muitos tipos de subclasse. Isso é permitido porque cada objeto de subclasse *é um* objeto de sua superclasse. Por exemplo, podemos atribuir a referência de um objeto `BasePlusCommissionEmployee` a uma variável da superclasse `CommissionEmployee`, pois um `BasePlusCommissionEmployee` *é um* `CommissionEmployee` – podemos tratar um `BasePlusCommissionEmployee` como um `CommissionEmployee`.

Conforme você vai aprender mais adiante neste apêndice, *não é permitido tratar um objeto de superclasse como um objeto de subclasse*, pois um objeto de superclasse *não é* um objeto de qualquer uma de suas subclasses. Por exemplo, não podemos atribuir a referência de um objeto `CommissionEmployee` a uma variável da subclasse `BasePlusCommissionEmployee`, pois um `CommissionEmployee` *não é* um `BasePlusCommissionEmployee` – um `CommissionEmployee` *não* tem uma variável de instância `baseSalary` e *não* tem métodos `setBaseSalary` e `getBaseSalary`. A relação *é um* se aplica apenas *para cima na hierarquia*, de uma subclasse para suas superclasses diretas e *indiretas*, e *não* vice-versa (isto é, *não para baixo na hierarquia*, de uma superclasse para suas subclasses).

O compilador Java *permite* a atribuição de uma referência de superclasse para uma variável de subclasse se *convertermos* explicitamente a referência de superclasse para o tipo da subclasse – uma técnica que discutimos na seção G.10. Por que desejaríamos fazer tal atribuição? Uma referência de superclasse só pode ser usada para chamar os métodos declarados na superclasse – chamar métodos exclusivos da subclasse por meio de uma referência de superclasse resulta em erros de compilação. Se um programa precisa efetuar uma operação específica da subclasse em um objeto de subclasse referenciado por uma variável de superclasse, ele deve primeiro converter a referência de superclasse em uma referência de subclasse, por meio de uma técnica conhecida como ***downcasting***. Isso permite que o programa chame métodos de subclasse que *não* estão na superclasse. Mostramos um exemplo de *downcasting* na seção G.10.

O exemplo da Fig. G.13 demonstra três maneiras de usar variáveis de superclasse e subclasse para armazenar referências para objetos de superclasse e subclasse. As duas primeiras são simples – como na seção G.4, atribuímos uma referência de superclasse a uma variável de superclasse e uma referência de subclasse a uma variável de subclasse. Então, demonstramos a relação entre subclasses e superclasses (isto é, a relação *é um*), atribuindo uma referência de subclasse a uma variável de superclasse. Esse programa usa as classes `CommissionEmployee` e `BasePlusCommissionEmployee` da Fig. G.10 e da Fig. G.11, respectivamente.

```

1 // Fig. G.13: PolymorphismTest.java
2 // Atribuindo referências de superclasse e subclasse a
3 // variáveis de superclasse e subclasse.
4
5 public class PolymorphismTest
6 {
7 public static void main(String[] args)
8 {
9 // atribui referência de superclasse à variável de superclasse
10 CommissionEmployee commissionEmployee = new CommissionEmployee(
11 "Sue", "Jones", "222-22-2222", 10000, .06);
12
13 // atribui referência de subclasse à variável de subclasse
14 BasePlusCommissionEmployee basePlusCommissionEmployee =
15 new BasePlusCommissionEmployee(
16 "Bob", "Lewis", "333-33-3333", 5000, .04, 300);
17
18 // chama toString no objeto de superclasse usando variável de superclasse
19 System.out.printf("%s %s:\n\n%s\n\n",
20 "Call CommissionEmployee's toString with superclass reference ",
21 "to superclass object", commissionEmployee.toString());
22
23 // chama toString no objeto de subclasse usando variável de subclasse
24 System.out.printf("%s %s:\n\n%s\n\n",
25 "Call BasePlusCommissionEmployee's toString with subclass",
26 "reference to subclass object",
27 basePlusCommissionEmployee.toString());
28
29 // chama toString no objeto de subclasse usando variável de superclasse
30 CommissionEmployee commissionEmployee2 =
31 basePlusCommissionEmployee;
32 System.out.printf("%s %s:\n\n%s\n\n",
33 "Call BasePlusCommissionEmployee's toString with superclass",
34 "reference to subclass object", commissionEmployee2.toString());
35 } // fim de main
36 } // fim da classe PolymorphismTest

```

Call CommissionEmployee's toString with superclass reference to superclass object:

commission employee: Sue Jones  
social security number: 222-22-2222  
gross sales: 10000.00  
commission rate: 0.06

Call BasePlusCommissionEmployee's toString with subclass reference to subclass object:

base-salaried commission employee: Bob Lewis  
social security number: 333-33-3333  
gross sales: 5000.00  
commission rate: 0.04  
base salary: 300.00

Call BasePlusCommissionEmployee's toString with superclass reference to subclass object:

base-salaried commission employee: Bob Lewis  
social security number: 333-33-3333  
gross sales: 5000.00  
commission rate: 0.04  
base salary: 300.00

**Figura G.13** Atribuindo referências de superclasse e subclasse a variáveis de superclasse e subclasse.

Na Fig. G.13, as linhas 10 e 11 criam um objeto `CommissionEmployee` e atribuem sua referência a uma variável de `CommissionEmployee`. As linhas 14 a 16 criam um objeto `BasePlusCommissionEmployee` e atribuem sua referência a uma variável de `BasePlusCommissionEmployee`. Essas atribuições são naturais – por exemplo, a principal finalidade de uma variável de `CommissionEmployee` é armazenar uma referência para um objeto `CommissionEmployee`. As linhas 19 a 21 usam `commissionEmployee` para chamar `toString` explicitamente. Como `commissionEmployee` se refere a um objeto `CommissionEmployee`, é chamada a versão de `toString` da superclasse `CommissionEmployee`. Do mesmo modo, as linhas 24 a 27 usam `basePlusCommissionEmployee` para chamar `toString` explicitamente no objeto `BasePlusCommissionEmployee`. Isso chama a versão de `toString` da subclasse `BasePlusCommissionEmployee`. Então, as linhas 30 e 31 atribuem a referência do objeto de subclasse `basePlusCommissionEmployee` a uma variável da superclasse `CommissionEmployee`, a qual as linhas 32 a 34 usam para chamar o método `toString`. *Quando uma variável de superclasse contém uma referência para um objeto de subclasse e essa referência é usada para chamar um método, é chamada a versão da subclasse do método.* Assim, `commissionEmployee2.toString()` na linha 34 chama na verdade o método `toString` da classe `BasePlusCommissionEmployee`. O compilador Java permite esse “cruzamento” porque um objeto de uma subclasse é um objeto de sua superclasse (mas não vice-versa). Quando o compilador encontra uma chamada de método feita por meio de uma variável, ele determina se o método pode ser chamado verificando o tipo de classe da variável. Se essa classe contém a declaração de método correta (ou herda uma), a chamada é compilada. No momento da execução, o tipo do objeto ao qual a variável se refere determina o método realmente usado. Esse processo, denominado *vinculação dinâmica*, está discutido em detalhes na seção G.10.

## G.9 Classes e métodos abstratos

Quando pensamos em uma classe, presumimos que os programas vão criar objetos desse tipo. Às vezes, é interessante declarar classes – denominadas **classes abstratas** – para as quais você *nunca* pretende criar objetos. Como elas são usadas apenas como superclasses nas hierarquias de herança, nos referimos a elas como **superclasses abstratas**. Essas classes não podem ser usadas para instanciar objetos, pois, como veremos em breve, as classes abstratas são *incompletas*. As subclasses devem declarar as “partes ausentes” para se tornarem classes “concretas”, a partir das quais você *pode* instanciar objetos. Caso contrário, essas subclasses também serão abstratas. Demonstramos as classes abstratas na seção G.10.

### Finalidade das classes abstratas

A finalidade de uma classe abstrata é fornecer uma superclasse apropriada a partir da qual outras classes podem herdar e, assim, compartilhar um projeto comum. Na hierarquia Forma da Fig. G.3, por exemplo, as subclasses herdam a noção do que significa ser uma Forma – talvez atributos comuns, como `localização`, `cor` e `espessuraDaBorda`, e comportamentos como `desenhar`, `mover`, `redimensionar` e `mudarCor`. As classes que podem ser usadas para instanciar objetos são denominadas **classes concretas**. Essas classes fornecem implementações de *cada* método que declaram (algumas das implementações podem ser herdadas). Por exemplo, poderíamos produzir as classes concretas `Círculo`, `Quadrado` e `Triângulo` a partir da superclasse abstrata `FormaBidimensional`. Do mesmo modo, poderíamos produzir as classes concretas `Esfera`, `Cubo` e `Tetraedro` a partir da superclasse abstrata `FormaTridimensional`. As superclasses abstratas são *gerais demais* para criar objetos reais – elas especificam apenas o que é *comum* entre as subclasses. Precisamos ser mais *específicos* antes de podermos criar objetos. Por exemplo, se você envia a



mensagem desenhar para a classe abstrata `FormaBidimensional`, a classe sabe que formas bidimensionais devem ser *desenhadas*, mas não sabe a forma *específica* a desenhar; portanto, não pode implementar um método desenhar real. As classes concretas fornecem os *detalhes específicos* que as tornam adequadas para instanciar objetos.

Nem todas as hierarquias contêm classes abstratas. No entanto, muitas vezes você vai escrever código cliente que utiliza apenas tipos de superclasse abstrata para reduzir as dependências do código a um intervalo de tipos de subclasse. Por exemplo, você pode escrever um método com um parâmetro de um tipo de superclasse abstrata. Quando chamado, esse método pode receber um objeto de *qualquer* classe concreta que estenda direta ou indiretamente a superclasse especificada como tipo do parâmetro.

Às vezes, as classes abstratas constituem vários níveis de uma hierarquia. Por exemplo, a hierarquia `Forma` da Fig. G.3 começa com a classe abstrata `Forma`. No nível seguinte da hierarquia estão as classes *abstratas* `FormaBidimensional` e `FormaTridimensional`. O próximo nível declara classes *concretas* para `FormasBidimensionais` (`Círculo`, `Quadrado` e `Triângulo`) e para `FormasTridimensionais` (`Esfera`, `Cubo` e `Tetraedro`).

### Declarando uma classe abstrata e métodos abstratos

Você torna uma classe abstrata declarando-a com a palavra-chave **abstract**. Normalmente, uma classe abstrata contém um ou mais **métodos abstratos**. Um método abstrato é o que tem a palavra-chave `abstract` em sua declaração, como em

```
public abstract void draw(); // método abstrato
```

Os métodos abstratos *não* fornecem implementações. Uma classe que contém *quaisquer* métodos abstratos deve ser explicitamente declarada como `abstract`, mesmo que essa classe contenha alguns métodos concretos (não abstratos). Cada subclasse concreta de uma superclasse abstrata também deve fornecer implementações concretas de cada um dos métodos abstratos da superclasse. Construtores e métodos estáticos não podem ser declarados como `abstract`. Construtores não são herdados; portanto, um construtor abstrato nunca poderia ser implementado. Embora os métodos estáticos não *sejam* herdados, eles *não podem* ser sobrescritos. Como os métodos abstratos se destinam a ser sobrescritos para que possam processar objetos de acordo com seus tipos, não faria sentido declarar um método estático como `abstract`.



#### Observação sobre engenharia de software G.6

*Uma classe abstrata declara atributos e comportamentos comuns (tanto abstratos como concretos) das várias classes de uma hierarquia de classes. Normalmente, uma classe abstrata contém um ou mais métodos abstratos que as subclasses devem sobrescrever, caso devam ser concretas. As variáveis de instância e os métodos concretos de uma classe abstrata estão sujeitos às regras normais da herança.*

### Usando classes abstratas para declarar variáveis

Embora não possamos instanciar objetos de superclasses abstratas, em breve você vai ver que *podemos* usar superclasses abstratas para declarar variáveis que podem armazenar referências para objetos de qualquer classe concreta derivada dessas superclasses abstratas. Normalmente, os programas usam tais variáveis para manipular objetos de subclasse de forma polimórfica. Também é possível usar nomes de superclasses abstratas para chamar métodos estáticos declarados nessas superclasses.

Considere outra aplicação de polimorfismo. Um programa de desenho precisa exibir muitas formas, incluindo tipos de novas formas que você vai adicionar ao sistema



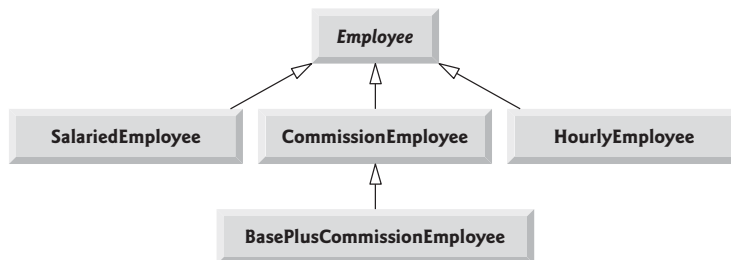
depois de escrever o programa de desenho. Talvez esse programa precise exibir formas como Círculos, Triângulos, Retângulos ou outras que derivem da classe abstrata Forma. O programa de desenho usa variáveis de Forma para gerenciar os objetos que são exibidos. Para desenhar qualquer objeto dessa hierarquia de herança, o programa de desenho usa uma variável da superclasse Forma contendo uma referência para o objeto de subclasse, a fim de chamar o método desenhar do objeto. Esse método é declarado como abstract na superclasse Forma; portanto, cada subclasse concreta *deve* implementar o método desenhar de uma maneira *específica* para essa forma – cada objeto da hierarquia de herança Forma *sabe como se desenhar*. O programa de desenho não precisa se preocupar com o tipo de cada objeto ou se encontrou objetos desse tipo.

## G.10 Estudo de caso: sistema de folha de pagamento usando polimorfismo

Esta seção examina novamente a hierarquia `CommissionEmployee-BasePlusCommissionEmployee` que exploramos ao longo da seção G.4. Agora, usamos um método abstrato e polimorfismo para efetuar cálculos de folha de pagamento com base em uma hierarquia de herança de funcionários melhorada que satisfaz os seguintes requisitos:

*Uma empresa faz pagamentos semanais aos seus funcionários. Os funcionários são de quatro tipos: os assalariados recebem um salário semanal fixo, independentemente do número de horas trabalhadas; os funcionários remunerados por hora recebem a hora trabalhada e horas extras (isto é, 1,5 vez o valor da hora) por todas as horas trabalhadas que ultrapassarem 40 horas; os funcionários comissionados recebem uma porcentagem de suas vendas; e os funcionários assalariados com comissão recebem um salário-base mais uma porcentagem de suas vendas. Para o período de pagamento em curso, a empresa decidiu recompensar os funcionários assalariados com comissão, adicionando 10% aos seus salários-base. A empresa quer escrever um aplicativo que efetue seus cálculos de folha de pagamento de forma polimórfica.*

Usamos a classe abstract `Employee` para representar a noção geral de funcionário. As classes que estendem `Employee` são `SalariedEmployee`, `CommissionEmployee` e `HourlyEmployee`. A classe `BasePlusCommissionEmployee` – que estende `CommissionEmployee` – representa o último tipo de funcionário. O diagrama de classe em UML da Fig. G.14 mostra a hierarquia de herança de nosso aplicativo polimórfico de folha de pagamento de funcionário. O nome da classe abstrata `Employee` está em itálico – uma convenção da UML.



**Figura G.14** Diagrama de classe em UML da hierarquia `Employee`.

A superclasse abstrata `Employee` declara a “interface” da hierarquia – ou seja, o conjunto de métodos que um programa pode chamar em todos os objetos `Employee`. Usamos o termo “interface” de forma geral aqui para nos referirmos às várias maneiras pelas quais os programas podem se comunicar com os objetos de qualquer subclasse de `Employee`. Cuidado para não confundir a noção geral de “interface” com a noção formal de interface Java, o tema da seção G.12. Cada funcionário, independentemente da maneira como seus ganhos são calculados, tem um nome, um sobrenome e um número de previdência social, de modo que as variáveis de instância `private firstName`, `lastName` e `socialSecurityNumber` aparecem na superclasse abstrata `Employee`. As seções a seguir implementam a hierarquia de classes de `Employee` da Fig. G.14. A primeira seção implementa a superclasse abstrata `Employee`. Cada uma das quatro seções seguintes implementa uma das classes concretas. A última seção implementa um programa de teste que constrói objetos de todas essas classes e os processa de forma polimórfica.

### G.10.1 Superclasse abstrata `Employee`

A classe `Employee` (Fig. G.16) fornece os métodos `earnings` e `toString`, além dos métodos `get` e `set` que manipulam as variáveis de instância de `Employee`. Um método `earnings` certamente se aplica genericamente a todos os funcionários. Mas cada cálculo de ganho depende da classe do funcionário. Assim, declaramos `earnings` como `abstract` na superclasse `Employee`, pois uma implementação padrão não faz sentido para esse método – não há informações suficientes para determinar o valor a ser retornado por `earnings`. Cada subclasse sobrescreve `earnings` com uma implementação adequada. Para calcular os ganhos de um funcionário, o programa atribui a uma variável da superclasse `Employee` uma referência ao objeto do funcionário e, então, chama o método `earnings` nessa variável. Mantemos um array de variáveis de `Employee`, cada uma armazenando uma referência para um objeto `Employee`. (Evidentemente, não pode haver objetos `Employee`, pois `Employee` é uma classe abstrata. Contudo, graças à herança, todos os objetos de todas as subclasses de `Employee` podem ser considerados objetos `Employee`.) O programa vai iterar pelo array e chamar o método `earnings` para cada objeto `Employee`. A linguagem Java processa essas chamadas de método de forma polimórfica. Declarar `earnings` como um método `abstract` em `Employee` permite que as chamadas para `earnings` por meio de variáveis de `Employee` compilem, e obriga a cada subclasse direta e concreta de `Employee` sobrescrever `earnings`.

O método `toString` da classe `Employee` retorna uma `String` contendo o nome, sobrenome e número da previdência social do funcionário. Conforme veremos, cada subclasse de `Employee` sobrescreve o método `toString` para criar uma representação de `String` de um objeto dessa classe contendo o tipo do funcionário (por exemplo, “salaried employee:”), seguido pelo restante das informações do funcionário.

O diagrama da Fig. G.15 mostra cada uma das cinco classes da hierarquia no lado esquerdo e os métodos `earnings` e `toString` na parte superior. Para cada classe, o diagrama mostra os resultados desejados de cada método. Não listamos os métodos `get` e `set` da superclasse `Employee` porque não são sobrescritos em nenhuma das subclasses – cada um desses métodos é herdado e utilizado “no estado em que se encontra” por cada subclasse.

|                                      | earnings                                                                                                                          | toString                                                                                                                                                                                                                |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Employee                             | abstract                                                                                                                          | <i>firstName lastName</i><br>social security number: <i>SSN</i>                                                                                                                                                         |
| Salaried-<br>Employee                | weeklySalary                                                                                                                      | salaried employee: <i>firstName lastName</i><br>social security number: <i>SSN</i><br>weekly salary: <i>weeklySalary</i>                                                                                                |
| Hourly-<br>Employee                  | <pre> if (hours &lt;= 40)     wage * hours else if (hours &gt; 40) {     40 * wage +     ( hours - 40 ) *     wage * 1.5 } </pre> | hourly employee: <i>firstName lastName</i><br>social security number: <i>SSN</i><br>hourly wage: <i>wage</i> ; hours worked: <i>hours</i>                                                                               |
| Commission-<br>Employee              | commissionRate *<br>grossSales                                                                                                    | commission employee: <i>firstName lastName</i><br>social security number: <i>SSN</i><br>gross sales: <i>grossSales</i> ;<br>commission rate: <i>commissionRate</i>                                                      |
| BasePlus-<br>Commission-<br>Employee | (commissionRate *<br>grossSales) +<br>baseSalary                                                                                  | base salaried commission employee:<br><i>firstName lastName</i><br>social security number: <i>SSN</i><br>gross sales: <i>grossSales</i> ;<br>commission rate: <i>commissionRate</i> ;<br>base salary: <i>baseSalary</i> |

**Figura G.15** Interface polimórfica para as classes da hierarquia `Employee`.

Vamos considerar a declaração da classe `Employee` (Fig. G.16). A classe contém um construtor que recebe como argumentos o nome, sobrenome e número da previdência social (linhas 11 a 16); métodos *get* que retornam o nome, sobrenome e número da previdência social (linhas 25 a 28, 37 a 40 e 49 a 52, respectivamente); métodos *set* que configuram o nome, sobrenome e número da previdência social (linhas 19 a 22, 31 a 34 e 43 a 46, respectivamente); o método `toString` (linhas 55 a 60) que retorna a representação de `String` de um objeto `Employee`; e o método `abstract earnings` (linha 63), o qual será implementado por cada uma das subclasses concretas. O construtor de `Employee` não valida seus parâmetros neste exemplo; normalmente, essa validação deve ser fornecida.

```

1 // Fig. G.16: Employee.java
2 // Superclasse abstrata Employee.
3
4 public abstract class Employee
5 {
6 private String firstName;
7 private String lastName;
8 private String socialSecurityNumber;
9
10 // construtor de três argumentos
11 public Employee(String first, String last, String ssn)

```

**Figura G.16** Superclasse abstrata `Employee`. (continua)

```

12 {
13 firstName = first;
14 lastName = last;
15 socialSecurityNumber = ssn;s
16 } // fim do construtor de três argumentos de Employee
17
18 // configura o nome
19 public void setFirstName(String first)
20 {
21 firstName = first; // deve validar
22 } // fim do método setFirstName
23
24 // retorna o nome
25 public String getFirstName()
26 {
27 return firstName;
28 } // fim do método getFirstName
29
30 // configura o sobrenome
31 public void setLastName(String last)
32 {
33 lastName = last; // deve validar
34 } // fim do método setLastName
35
36 // retorna o sobrenome
37 public String getLastName()
38 {
39 return lastName;
40 } // fim do método getLastName
41
42 // configura o número da previdência social
43 public void setSocialSecurityNumber(String ssn)
44 {
45 socialSecurityNumber = ssn; // deve validar
46 } // fim do método setSocialSecurityNumber
47
48 // retorna o número da previdência social
49 public String getSocialSecurityNumber()
50 {
51 return socialSecurityNumber;
52 } // fim do método getSocialSecurityNumber
53
54 // retorna representação de String do objeto Employee
55 @Override
56 public String toString()
57 {
58 return String.format("%s %s\nsocial security number: %s",
59 getFirstName(), getLastName(), getSocialSecurityNumber());
60 } // fim do método toString
61
62 // método abstrato sobrescrito pelas subclasses concretas
63 public abstract double earnings(); // nenhuma implementação aqui
64 } // fim da classe abstrata Employee

```

**Figura G.16** Superclasse abstrata Employee.

Por que decidimos declarar `earnings` como um método `abstract`? Simplesmente não faz sentido fornecer uma implementação desse método na classe `Employee`. Não podemos calcular os ganhos de um `Employee` *geral* – precisamos primeiro saber o tipo *específico* de `Employee` para determinar o cálculo de ganhos apropriado. Declarando esse método como `abstract`, indicamos que cada subclasse concreta *deve* fornecer uma im-

plementação de earnings apropriada e que um programa poderá usar variáveis da superclasse `Employee` para chamar o método `earnings` de forma polimórfica para qualquer tipo de `Employee`.

### G.10.2 Subclasse concreta `SalariedEmployee`

A classe `SalariedEmployee` (Fig. G.17) estende a classe `Employee` (linha 4) e sobreescreve o método abstrato `earnings` (linhas 33 a 37), o que torna `SalariedEmployee` uma classe concreta. A classe contém um construtor (linhas 9 a 14) que recebe como argumentos um nome, um sobrenome, um número de previdência social e um salário semanal; um método `set` para atribuir um novo valor não negativo à variável de instância `weeklySalary` (linhas 17 a 24); um método `get` para retornar o valor de `weeklySalary` (linhas 27 a 30); um método `earnings` (linhas 33 a 37) para calcular os ganhos de um `SalariedEmployee`; e um método `toString` (linhas 40 a 45), o qual retorna uma `String` que inclui o tipo do funcionário, ou seja, "salaried employee: ", seguido das informações específicas do funcionário, produzidas pelo método `toString` da superclasse `Employee` e pelo método `getWeeklySalary` de `SalariedEmployee`. O construtor da classe `SalariedEmployee` passa o nome, sobrenome e número da previdência social para o construtor de `Employee` (linha 12) a fim de inicializar as variáveis de instância `private` não herdadas da superclasse. O método `earnings` sobreescreve o método abstrato `earnings` de `Employee` para fornecer uma implementação concreta que retorna o salário semanal do `SalariedEmployee`. Se não implementarmos `earnings`, a classe `SalariedEmployee` deverá ser declarada `abstract` – caso contrário, a classe `SalariedEmployee` não compilará. Evidentemente, queremos que `SalariedEmployee` seja uma classe concreta neste exemplo.

O método `toString` (linhas 40 a 45) sobreescreve o método `toString` de `Employee`. Se a classe `SalariedEmployee` não sobreescrevesse `toString`, teria herdado a versão de `Employee` de `toString`. Nesse caso, o método `toString` de `SalariedEmployee` simplesmente retornaria o nome completo e o número da previdência social do funcionário, o que não representa adequadamente um objeto `SalariedEmployee`. Para produzir uma representação de `String` completa de um `SalariedEmployee`, o método `toString` da subclasse retorna "salaried employee: ", seguido das informações específicas da superclasse `Employee` (isto é, nome, sobrenome e número da previdência social), obtidas pela chamada do método `toString` da superclasse (linha 44) – esse é um excelente exemplo de reutilização de código. A representação de `String` de um `SalariedEmployee` também contém o salário semanal do funcionário, obtido pela chamada ao método `getWeeklySalary` da classe.

```

1 // Fig. G.17: SalariedEmployee.java
2 // A classe concreta SalariedEmployee estende a classe abstrata Employee.
3
4 public class SalariedEmployee extends Employee
5 {
6 private double weeklySalary;
7
8 // construtor de quatro argumentos
9 public SalariedEmployee(String first, String last, String ssn,
10 double salary)
11 {
12 super(first, last, ssn); // passa para o construtor de Employee
13 setWeeklySalary(salary); // valida e armazena o salário

```

**Figura G.17** A classe concreta `SalariedEmployee` estende a classe abstrata `Employee`.

```

14 } // fim do construtor de quatro argumentos de SalariedEmployee
15
16 // configura o salário
17 public void setWeeklySalary(double salary)
18 {
19 if (salary >= 0.0)
20 baseSalary = salary;
21 else
22 throw new IllegalArgumentException(
23 "Weekly salary must be >= 0.0");
24 } // fim do método setWeeklySalary
25
26 // retorna o salário
27 public double getWeeklySalary()
28 {
29 return weeklySalary;
30 } // fim do método getWeeklySalary
31
32 // calcula os ganhos; sobrescreve o método abstrato earnings em Employee
33 @Override
34 public double earnings()
35 {
36 return getWeeklySalary();
37 } // fim do método earnings
38
39 // retorna representação de String do objeto SalariedEmployee
40 @Override
41 public String toString()
42 {
43 return String.format("salaried employee: %s\n%s: $%,.2f",
44 super.toString(), "weekly salary", getWeeklySalary());
45 } // fim do método toString
46 } // fim da classe SalariedEmployee

```

**Figura G.17** A classe concreta `SalariedEmployee` estende a classe abstrata `Employee`.

### G.10.3 Subclasse concreta `HourlyEmployee`

A classe `HourlyEmployee` (Fig. G.18) também estende `Employee` (linha 4). A classe contém um construtor (linhas 10 a 16) que recebe como argumentos um nome, um sobrenome, um número de previdência social, uma remuneração horária e o número de horas trabalhadas. As linhas 19 a 26 e 35 a 42 declaram métodos *set* que atribuem novos valores às variáveis de instância `wage` e `hours`, respectivamente. O método `setWage` (linhas 19 a 26) garante que `wage` não seja negativo, e o método `setHours` (linhas 35 a 42) garante que `hours` esteja entre 0 e 168 (o número total de horas em uma semana) inclusive. A classe `HourlyEmployee` inclui também métodos *get* (linhas 29 a 32 e 45 a 48) para retornar os valores de `wage` e `hours`, respectivamente; um método `earnings` (linhas 51 a 58) para calcular os ganhos de um `HourlyEmployee`; e um método `toString` (linhas 61 a 67), o qual retorna uma `String` contendo o tipo do funcionário ("hourly employee: ") e informações específicas do funcionário. Assim como o construtor de `SalariedEmployee`, o construtor de `HourlyEmployee` passa o nome, sobrenome e número da previdência social para o construtor da superclasse `Employee` (linha 13) a fim de inicializar as variáveis de instância `private`. Além disso, o método `toString` chama o método `toString` da superclasse (linha 65) para obter as informações específicas do `Employee` (isto é, nome, sobrenome e número da previdência social) – esse é outro ótimo exemplo de reutilização de código.

```

1 // Fig. G.18: HourlyEmployee.java
2 // A classe HourlyEmployee estende Employee.
3
4 public class HourlyEmployee extends Employee
5 {
6 private double wage; // remuneração por hora
7 private double hours; // horas trabalhadas por semana
8
9 // construtor de cinco argumentos
10 public HourlyEmployee(String first, String last, String ssn,
11 double hourlyWage, double hoursWorked)
12 {
13 super(first, last, ssn);
14 setWage(hourlyWage); // valida a remuneração horária
15 setHours(hoursWorked); // valida as horas trabalhadas
16 } // fim do construtor de cinco argumentos de HourlyEmployee
17
18 // configura a remuneração
19 public void setWage(double hourlyWage)
20 {
21 if (hourlyWage >= 0.0)
22 wage = hourlyWage;
23 else
24 throw new IllegalArgumentException(
25 "Hourly wage must be >= 0.0");
26 } // fim do método setWage
27
28 // retorna a remuneração
29 public double getWage()
30 {
31 return wage;
32 } // fim do método getWage
33
34 // configura as horas trabalhadas
35 public void setHours(double hoursWorked)
36 {
37 if ((hoursWorked >= 0.0) && (hoursWorked <= 168.0))
38 hours = hoursWorked;
39 else
40 throw new IllegalArgumentException(
41 "Hours worked must be >= 0.0 and <= 168.0");
42 } // fim do método setHours
43
44 // retorna as horas trabalhadas
45 public double getHours()
46 {
47 return hours;
48 } // fim do método getHours
49
50 // calcula os ganhos; sobrescreve o método abstrato earnings em Employee
51 @Override
52 public double earnings()
53 {
54 if (getHours() <= 40) // nenhuma hora extra
55 return getWage() * getHours();
56 else
57 return 40 * getWage() + (getHours() - 40) * getWage() * 1.5;
58 } // fim do método earnings
59
60 // retorna representação de String do objeto HourlyEmployee
61 @Override

```

**Figura G.18** A classe HourlyEmployee estende Employee. (continua)

```

62 public String toString()
63 {
64 return String.format("hourly employee: %s\n%s: $%,.2f; %s: $%,.2f",
65 super.toString(), "hourly wage", getWage(),
66 "hours worked", getHours());
67 } // fim do método toString
68 } // fim da classe HourlyEmployee

```

**Figura G.18** A classe `HourlyEmployee` estende `Employee`.

#### G.10.4 Subclasse concreta `CommissionEmployee`

A classe `CommissionEmployee` (Fig. G.19) estende a classe `Employee` (linha 4). A classe contém um construtor (linhas 10 a 16) que recebe um nome, um sobrenome, um número de previdência social, um valor de vendas e uma taxa de comissão; métodos *set* (linhas 19 a 26 e 35 a 42) para atribuir novos valores às variáveis de instância `commissionRate` e `grossSales`, respectivamente; métodos *get* (linhas 29 a 32 e 45 a 48) que recuperam os valores dessas variáveis de instância; método `earnings` (linhas 51 a 55) para calcular os ganhos de um `CommissionEmployee`; e método `toString` (linhas 58 a 65), o qual retorna o tipo do funcionário, a saber, "commission employee: ", e informações específicas do funcionário. O construtor também passa o nome, sobrenome e número da previdência social para o construtor de `Employee` (linha 13) a fim de inicializar as variáveis de instância *private* de `Employee`. O método `toString` chama o método `toString` da superclasse (linha 62) para obter as informações específicas do `Employee` (isto é, nome, sobrenome e número da previdência social).

```

1 // Fig. G.19: CommissionEmployee.java
2 // A classe CommissionEmployee estende Employee.
3
4 public class CommissionEmployee extends Employee
5 {
6 private double grossSales; // vendas brutas semanais
7 private double commissionRate; // porcentagem de comissão
8
9 // construtor de cinco argumentos
10 public CommissionEmployee(String first, String last, String ssn,
11 double sales, double rate)
12 {
13 super(first, last, ssn);
14 setGrossSales(sales);
15 setCommissionRate(rate);
16 } // fim do construtor de cinco argumentos de CommissionEmployee
17
18 // configura a taxa de comissão
19 public void setCommissionRate(double rate)
20 {
21 if (rate > 0.0 && rate < 1.0)
22 commissionRate = rate;
23 else
24 throw new IllegalArgumentException(
25 "Commission rate must be > 0.0 and < 1.0");
26 } // fim do método setCommissionRate
27
28 // retorna a taxa de comissão
29 public double getCommissionRate()
30 {

```

**Figura G.19** A classe `CommissionEmployee` estende `Employee`. (continua)



```

31 return commissionRate;
32 } // fim do método getCommissionRate
33
34 // configura o valor das vendas brutas
35 public void setGrossSales(double sales)
36 {
37 if (sales >= 0.0)
38 grossSales = sales;
39 else
40 throw new IllegalArgumentException(
41 "Gross sales must be >= 0.0");
42 } // fim do método setGrossSales
43
44 // retorna o valor das vendas brutas
45 public double getGrossSales()
46 {
47 return grossSales;
48 } // fim do método getGrossSales
49
50 // calcula os ganhos; sobrescreve o método abstrato earnings em Employee
51 @Override
52 public double earnings()
53 {
54 return getCommissionRate() * getGrossSales();
55 } // fim do método earnings
56
57 // retorna representação de String do objeto CommissionEmployee
58 @Override
59 public String toString()
60 {
61 return String.format("%s: %s\n%s: $%,.2f; %s: %.2f",
62 "commission employee", super.toString(),
63 "gross sales", getGrossSales(),
64 "commission rate", getCommissionRate());
65 } // fim do método toString
66 } // fim da classe CommissionEmployee

```

**Figura G.19** A classe `CommissionEmployee` estende `Employee`.

### G.10.5 Subclasse concreta indireta `BasePlusCommissionEmployee`

A classe `BasePlusCommissionEmployee` (Fig. G.20) estende a classe `CommissionEmployee` (linha 4) e, portanto, é uma subclasse *indireta* da classe `Employee`. A classe `BasePlusCommissionEmployee` tem um construtor (linhas 9 a 14) que recebe como argumentos um nome, um sobrenome, um número de previdência social, um valor de vendas, uma taxa de comissão e um salário-base. Então, ela passa todas essas informações, exceto o salário-base, para o construtor de `CommissionEmployee` (linha 12) a fim de inicializar os membros herdados. `BasePlusCommissionEmployee` contém também um método *set* (linhas 17 a 24) para atribuir um novo valor à variável de instância `baseSalary` e um método *get* (linhas 27 a 30) para retornar o valor de `baseSalary`. O método `earnings` (linhas 33 a 37) calcula os ganhos de um `BasePlusCommissionEmployee`. A linha 36 no método `earnings` chama o método `earnings` da superclasse `CommissionEmployee` para calcular a parte da comissão dos ganhos do funcionário – esse é outro bom exemplo de reutilização de código. O método `toString` de `BasePlusCommissionEmployee` (linhas 40 a 46) cria uma representação de `String` de um `BasePlusCommissionEmployee` contendo "base-salaried", seguido

da String obtida pela chamada ao método `toString` da superclasse `CommissionEmployee` (outro exemplo de reutilização de código) e pelo salário-base. O resultado é uma String começando com "base-salaried commission employee", seguido do restante das informações do `BasePlusCommissionEmployee`. Lembre-se de que o método `toString` de `CommissionEmployee` obtém o nome, sobrenome e número da previdência social do funcionário chamando o método `toString` de sua superclasse (ou seja, `Employee`) – mais um exemplo de reutilização de código. O método `toString` de `BasePlusCommissionEmployee` inicia um encadeamento de chamadas de método que abrange todos os três níveis da hierarquia `Employee`.

```

1 // Fig. G.20: BasePlusCommissionEmployee.java
2 // A classe BasePlusCommissionEmployee estende CommissionEmployee.
3
4 public class BasePlusCommissionEmployee extends CommissionEmployee
5 {
6 private double baseSalary; // salário-base por semana
7
8 // construtor de seis argumentos
9 public BasePlusCommissionEmployee(String first, String last,
10 String ssn, double sales, double rate, double salary)
11 {
12 super(first, last, ssn, sales, rate);
13 setBaseSalary(salary); // valida e armazena o salário-base
14 } // fim do construtor de seis argumentos de BasePlusCommissionEmployee
15
16 // configura o salário-base
17 public void setBaseSalary(double salary)
18 {
19 if (salary >= 0.0)
20 baseSalary = salary;
21 else
22 throw new IllegalArgumentException(
23 "Base salary must be >= 0.0");
24 } // fim do método setBaseSalary
25
26 // retorna o salário-base
27 public double getBaseSalary()
28 {
29 return baseSalary;
30 } // fim do método getBaseSalary
31
32 // calcula os ganhos; sobrescreve o método abstrato earnings em CommissionEmployee
33 @Override
34 public double earnings()
35 {
36 return getBaseSalary() + super.earnings();
37 } // fim do método earnings
38
39 // retorna representação de String do objeto BasePlusCommissionEmployee
40 @Override
41 public String toString()
42 {
43 return String.format("%s %s; %s: $%,.2f",
44 "base-salaried", super.toString(),
45 "base salary", getBaseSalary());
46 } // fim do método toString
47 } // fim da classe BasePlusCommissionEmployee

```

**Figura G.20** A classe `BasePlusCommissionEmployee` estende `CommissionEmployee`.

### G.10.6 Processamento polimórfico, operador instanceof e downcasting

Para testar nossa hierarquia `Employee`, o aplicativo da Fig. G.21 cria um objeto de cada uma das quatro classes concretas `SalariedEmployee`, `HourlyEmployee`, `CommissionEmployee` e `BasePlusCommissionEmployee`. O programa manipula esses objetos de forma não polimórfica por meio de variáveis do tipo próprio de cada objeto e, então, de forma polimórfica, usando um array de variáveis de `Employee`. Ao processar os objetos de forma polimórfica, o programa aumenta em 10% o salário-base de cada `BasePlusCommissionEmployee` – isso exige *determinar o tipo do objeto no momento da execução*. Por fim, o programa determina de forma polimórfica e gera o tipo de cada objeto do array `Employee`. As linhas 9 a 18 criam objetos de cada uma das quatro subclasses concretas de `Employee`. As linhas 22 a 30 geram a representação de `String` e os ganhos de cada um desses objetos de forma *não polimórfica*. O método `toString` de cada objeto é chamado *implicitamente* por `printf`, quando o objeto é gerado como uma `String` com o especificador de formato `%s`.

```

1 // Fig. G.21: PayrollSystemTest.java
2 // Programa de teste da hierarquia Employee.
3
4 public class PayrollSystemTest
5 {
6 public static void main(String[] args)
7 {
8 // cria objetos de subclasse
9 SalariedEmployee salariedEmployee =
10 new SalariedEmployee("John", "Smith", "111-11-1111", 800.00);
11 HourlyEmployee hourlyEmployee =
12 new HourlyEmployee("Karen", "Price", "222-22-2222", 16.75, 40);
13 CommissionEmployee commissionEmployee =
14 new CommissionEmployee(
15 "Sue", "Jones", "333-33-3333", 10000, .06);
16 BasePlusCommissionEmployee basePlusCommissionEmployee =
17 new BasePlusCommissionEmployee(
18 "Bob", "Lewis", "444-44-4444", 5000, .04, 300);
19
20 System.out.println("Employees processed individually:\n");
21
22 System.out.printf("%s\n%s: $%,.2f\n\n",
23 salariedEmployee, "earned", salariedEmployee.earnings());
24 System.out.printf("%s\n%s: $%,.2f\n\n",
25 hourlyEmployee, "earned", hourlyEmployee.earnings());
26 System.out.printf("%s\n%s: $%,.2f\n\n",
27 commissionEmployee, "earned", commissionEmployee.earnings());
28 System.out.printf("%s\n%s: $%,.2f\n\n",
29 basePlusCommissionEmployee,
30 "earned", basePlusCommissionEmployee.earnings());
31
32 // cria array Employee de quatro elementos
33 Employee[] employees = new Employee[4];
34
35 // inicializa o array com objetos Employee
36 employees[0] = salariedEmployee;
37 employees[1] = hourlyEmployee;
38 employees[2] = commissionEmployee;
39 employees[3] = basePlusCommissionEmployee;
40
41 System.out.println("Employees processed polymorphically:\n");

```

Figura G.21 Programa de teste da hierarquia `Employee`.

```

42
43 // processa genericamente cada elemento do array employees
44 for (Employee currentEmployee : employees)
45 {
46 System.out.println(currentEmployee); // chama toString
47
48 // determina se o elemento é um BasePlusCommissionEmployee
49 if (currentEmployee instanceof BasePlusCommissionEmployee)
50 {
51 // faz downcasting da referência de Employee para
52 // referência de BasePlusCommissionEmployee
53 BasePlusCommissionEmployee employee =
54 (BasePlusCommissionEmployee) currentEmployee;
55
56 employee.setBaseSalary(1.10 * employee.getBaseSalary());
57
58 System.out.printf(
59 "new base salary with 10%% increase is: $%,.2f\n",
60 employee.getBaseSalary());
61 } // fim do if
62
63 System.out.printf(
64 "earned $%,.2f\n\n", currentEmployee.earnings());
65 } // fim do for
66
67 // obtém o nome do tipo de cada objeto do array employees
68 for (int j = 0; j < employees.length; j++)
69 System.out.printf("Employee %d is a %s\n", j,
70 employees[j].getClass().getName());
71 } // fim de main
72 } // fim da classe PayrollSystemTest

```

Employees processed individually:

salaried employee: John Smith  
social security number: 111-11-1111  
weekly salary: \$800.00  
earned: \$800.00

hourly employee: Karen Price  
social security number: 222-22-2222  
hourly wage: \$16.75; hours worked: 40.00  
earned: \$670.00

commission employee: Sue Jones  
social security number: 333-33-3333  
gross sales: \$10,000.00; commission rate: 0.06  
earned: \$600.00

base-salaried commission employee: Bob Lewis  
social security number: 444-44-4444  
gross sales: \$5,000.00; commission rate: 0.04; base salary: \$300.00  
earned: \$500.00

Employees processed polymorphically:

salaried employee: John Smith  
social security number: 111-11-1111  
weekly salary: \$800.00  
earned \$800.00

**Figura G.21** Programa de teste da hierarquia Employee. (continua)

```

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
new base salary with 10% increase is: $330.00
earned $530.00

Employee 0 is a SalariedEmployee
Employee 1 is a HourlyEmployee
Employee 2 is a CommissionEmployee
Employee 3 is a BasePlusCommissionEmployee

```

**Figura G.21** Programa de teste da hierarquia Employee.

### ***Criando o array de objetos Employee***

A linha 33 declara `employees` e o atribui a um array de quatro variáveis de `Employee`. A linha 36 atribui a referência para um objeto `SalariedEmployee` a `employees[0]`. A linha 37 atribui a referência para um objeto `HourlyEmployee` a `employees[1]`. A linha 38 atribui a referência para um objeto `CommissionEmployee` a `employees[2]`. A linha 39 atribui a referência para um objeto `BasePlusCommissionEmployee` a `employees[3]`. Essas atribuições são permitidas porque um `SalariedEmployee` *é um* `Employee`, um `HourlyEmployee` *é um* `Employee`, um `CommissionEmployee` *é um* `Employee` e um `BasePlusCommissionEmployee` *é um* `Employee`. Portanto, podemos atribuir as referências de objetos `SalariedEmployee`, `HourlyEmployee`, `CommissionEmployee` e `BasePlusCommissionEmployee` a variáveis da superclasse `Employee`, *mesmo sendo Employee uma classe abstrata*.

### ***Processando objetos Employee de forma polimórfica***

As linhas 44 a 65 iteram pelo array `employees` e chamam os métodos `toString` e `earnings` com a variável `currentEmployee` de `Employee`, a qual recebe a referência para um objeto `Employee` diferente no array em cada iteração. A saída ilustra que são realmente chamados os métodos apropriados de cada classe. Todas as chamadas aos métodos `toString` e `earnings` são resolvidas no momento da execução, com base no tipo do objeto ao qual `currentEmployee` se refere. Esse processo é conhecido como **vinculação dinâmica** ou **vinculação tardia**. Por exemplo, a linha 46 chama *implicitamente* o método `toString` do objeto ao qual `currentEmployee` se refere. Como resultado da vinculação dinâmica, a linguagem Java decide qual o método `toString` da classe vai chamar *no momento da execução e não no momento da compilação*. Somente os métodos da classe `Employee` podem ser chamados por meio de uma variável de `Employee` (e `Employee`, é claro, inclui os métodos da classe `Object`). Uma referência de superclasse só pode ser usada para chamar os métodos da superclasse – as implementações de método de subclasse são chamadas de forma polimórfica.

### ***Efetuando operações específicas quanto ao tipo em BasePlusCommissionEmployees***

Executamos processamento especial em objetos `BasePlusCommissionEmployee` – à medida que os encontramos no momento da execução, aumentamos em 10% seus salários-base.

Ao processarmos objetos de forma polimórfica, normalmente não precisamos nos preocupar com os “detalhes específicos”, mas, para ajustar o salário-base, *precisamos* determinar o tipo específico de objeto `Employee` no momento da execução. A linha 49 usa o operador `instanceof` para determinar se o tipo de um objeto `Employee` em particular é `BasePlusCommissionEmployee`. A condição na linha 49 é verdadeira se o objeto referenciado por `currentEmployee` é um `BasePlusCommissionEmployee`. Isso também seria verdadeiro para qualquer objeto de uma subclasse de `BasePlusCommissionEmployee`, graças à relação *é um* que uma subclasse tem com sua superclasse. As linhas 53 e 54 fazem *downcasting* de `currentEmployee`, do tipo `Employee` para o tipo `BasePlusCommissionEmployee` — essa conversão só é permitida se o objeto tem uma relação *é um* com `BasePlusCommissionEmployee`. A condição na linha 49 garante que esse seja o caso. Essa conversão seria obrigatória se chamássemos os métodos `getBaseSalary` e `setBaseSalary` da subclasse `BasePlusCommissionEmployee` no objeto `Employee` atual — conforme você vai ver em breve, *chamar um método exclusivo de uma subclasse diretamente em uma referência de superclasse gera um erro de compilação*.



### Erro de programação comum G.3

*Atribuir uma variável de superclasse a uma variável de subclasse (sem uma conversão explícita) gera um erro de compilação.*



### Observação sobre engenharia de software G.7

*Se a referência de um objeto de subclasse foi atribuída a uma variável de uma de suas superclasses direta ou indireta no momento da execução, é aceitável fazer *downcasting* da referência armazenada nessa variável de superclasse para uma referência do tipo da subclasse. Antes de fazer essa conversão, use o operador `instanceof` para garantir que o objeto seja mesmo um objeto de uma subclasse apropriada.*



### Erro de programação comum G.4

*Ao se fazer *downcasting* de uma referência, ocorrerá uma exceção `ClassCastException` se o objeto referenciado no momento da execução não tiver uma relação *é um* com o tipo especificado no operador de conversão.*

Se a expressão com `instanceof` na linha 49 é `true`, as linhas 53 a 60 executam o processamento especial exigido para o objeto `BasePlusCommissionEmployee`. Usando a variável `employee` de `BasePlusCommissionEmployee`, a linha 56 chama os métodos `getBaseSalary` e `setBaseSalary` exclusivos da subclasse para recuperar e atualizar o salário-base do funcionário com o aumento de 10%.

### Chamando `earnings` de forma polimórfica

As linhas 63 e 64 chamam o método `earnings` em `currentEmployee`, o qual chama o método `earnings` do objeto da subclasse apropriada de forma polimórfica. Obter os ganhos de `SalariedEmployee`, `HourlyEmployee` e `CommissionEmployee` de forma polimórfica nas linhas 63 e 64 produz os mesmos resultados de obter os ganhos dos funcionários individualmente nas linhas 22 a 27. O valor dos ganhos obtidos para `BasePlusCommissionEmployee` nas linhas 63 e 64 é maior que o obtido nas linhas 28 e 30, devido ao aumento de 10% em seu salário-base.

### Usando reflexão para obter o nome de classe de cada objeto `Employee`

As linhas 68 a 70 exibem o tipo de cada funcionário como uma `String`, usando recursos básicos dos assim chamados recursos de reflexão da linguagem Java. Todo objeto conhece sua própria classe e pode acessar essa informação por meio do método `getClass`, o qual todas as classes herdam da classe `Object`. O método `getClass` retorna um objeto do tipo `Class` (do pacote `java.lang`), o qual contém informações sobre o tipo do objeto, incluindo seu nome de classe. A linha 70 chama `getClass` no objeto atual para obter sua classe de tempo de execução. O resultado da chamada de `getClass` é usado para chamar `getName` a fim de obter o nome de classe do objeto.

### Evitando erros de compilação com *downcasting*

No exemplo anterior, evitamos vários erros de compilação fazendo *downcasting* de uma variável de `Employee` para uma variável de `BasePlusCommissionEmployee`, nas linhas 53 e 54. Se você remover o operador de conversão (`BasePlusCommissionEmployee`) da linha 54 e tentar atribuir a variável `currentEmployee` de `Employee` diretamente à variável `employee` de `BasePlusCommissionEmployee`, receberá o erro de compilação "incompatible types". Esse erro indica que a tentativa de atribuir a referência do objeto de superclasse `currentEmployee` à variável de subclasse `employee` não é permitida. O compilador impede essa atribuição porque um `CommissionEmployee` não é um `BasePlusCommissionEmployee` — a relação é um *se aplica somente entre a subclasse e suas superclasses, não vice-versa*.

Do mesmo modo, se as linhas 56 e 60 usassem a variável de superclasse `currentEmployee` para chamar métodos exclusivos da subclasse `getBaseSalary` e `setBaseSalary`, receberíamos erros de compilação "cannot find symbol" nessas linhas. Não é permitido chamar métodos exclusivos da subclasse por meio de uma variável de superclasse — mesmo que as linhas 56 e 60 sejam executadas somente se `instanceof`, na linha 49, retornar `true` para indicar que `currentEmployee` armazena uma referência para um objeto `BasePlusCommissionEmployee`. Usando uma variável da superclasse `Employee`, só podemos chamar os métodos encontrados na classe `Employee` — `earnings`, `toString` e os métodos `get` e `set` de `Employee`.



#### Observação sobre engenharia de software G.8

*Embora o método realmente chamado dependa do tipo de runtime do objeto ao qual uma variável se refere, uma variável pode ser usada para chamar somente os métodos que são membros do tipo dessa variável, o qual é verificado pelo compilador.*

## G.10.7 Resumo das atribuições permitidas entre variáveis de superclasse e de subclasse

Agora que você já viu um aplicativo completo que processa diversos objetos de subclasse de forma polimórfica, resumimos o que pode e o que não pode ser feito com objetos e variáveis de superclasse e de subclasse. Embora um objeto de subclasse também *seja* um objeto da superclasse, ainda assim os dois são diferentes. Conforme discutido anteriormente, os objetos de subclasse podem ser tratados como objetos de suas superclasses. Mas, como a subclasse pode ter membros exclusivos adicionais, atribuir uma referência de superclasse a uma variável de subclasse não é permitido sem uma conversão explícita — tal atribuição deixaria os membros da subclasse indefinidos para o objeto da superclasse.

Discutimos quatro maneiras de atribuir referências de superclasse e de subclasse a variáveis de tipos da superclasse e da subclasse:

1. Atribuir uma referência de superclasse a uma variável de superclasse é natural.
2. Atribuir uma referência de subclasse a uma variável de subclasse é natural.
3. Atribuir uma referência de subclasse a uma variável de superclasse é seguro, pois o objeto da subclasse *é um* objeto de sua superclasse. Contudo, a variável de superclasse pode ser usada para se referir *apenas* a membros da superclasse. Se esse código faz referência a membros exclusivos da subclasse, por meio da variável de superclasse, o compilador acusa erros.
4. Atribuir uma referência de superclasse a uma variável de subclasse causa um erro de compilação. Para evitar esse erro, a referência de superclasse deve ser convertida explicitamente em um tipo de subclasse. No *momento da execução*, se o objeto ao qual a referência se refere *não* for um objeto de subclasse, ocorrerá uma exceção. (Para mais informações sobre tratamento de exceções, consulte o Apêndice H.) Você deve usar o operador `instanceof` para garantir que essa conversão seja feita somente se se tratar de um objeto de subclasse.

## G.11 Métodos e classes `final`

Vimos nas seções D.3 e D.10 que as variáveis podem ser declaradas como `final` para indicar que não podem ser modificadas após serem inicializadas – essas variáveis representam valores constantes. Também é possível declarar métodos, parâmetros de método e classes com o modificador `final`.

### **Métodos `final` não podem ser sobrescritos**

Um método `final` de uma superclasse *não pode* ser sobrescrito em uma subclasse – isso garante que a implementação do método `final` seja usada por todas as subclasses diretas e indiretas da hierarquia. Os métodos declarados como `private` são implicitamente `final`, pois não é possível sobrescrevê-los em uma subclasse. Os métodos declarados como `static` também são implicitamente `final`. A declaração de um método `final` nunca pode mudar; portanto, todas as subclasses utilizam a mesma implementação do método, e as chamadas aos métodos `final` são resolvidas em tempo de compilação – isso é conhecido como **vinculação estática**.

### **Classes `final` não podem ser superclasses**

Uma **classe `final`** declarada como `final` não pode ser uma superclasse (isto é, uma classe não pode estender uma classe `final`). Todos os métodos de uma classe `final` são implicitamente `final`. A classe `String` é um exemplo de classe `final`. Se fosse permitido criar uma subclasse de `String`, os objetos dessa subclasse poderiam ser usados sempre que `Strings` fossem esperadas. Como a classe `String` não pode ser estendida, os programas que utilizam `Strings` podem contar com a funcionalidade dos objetos `String` conforme especificada na API Java. Tornar a classe `final` também impede que os programadores criem subclasses que poderiam ignorar restrições de segurança. Para mais ideias sobre o uso da palavra-chave `final`, visite:

[docs.oracle.com/javase/tutorial/java/IandI/final.html](https://docs.oracle.com/javase/tutorial/java/IandI/final.html)

e

[www.ibm.com/developerworks/java/library/j-jtp1029.html](http://www.ibm.com/developerworks/java/library/j-jtp1029.html)





### Erro de programação comum G.5

*Declarar uma subclasse de uma classe `final` causa um erro de compilação.*



### Observação sobre engenharia de software G.9

*Na API Java, a ampla maioria das classes não é declarada como `final`. Isso possibilita a herança e o polimorfismo. Contudo, em alguns casos é importante declarar as classes como `final` – normalmente por questões de segurança.*

## G.12 Estudo de caso: criação e uso de interfaces

Nosso próximo exemplo (Figs. G.23 a G.27) examina novamente o sistema de folha de pagamento da seção G.10. Suponha que a empresa resolveu efetuar várias operações de contabilidade em um único aplicativo de contas a pagar – além de calcular os ganhos a serem pagos a cada funcionário, a empresa também precisa calcular os pagamentos de cada uma de várias faturas (isto é, cobranças por bens adquiridos). Embora sejam aplicadas a coisas não relacionadas (ou seja, funcionários e faturas), as duas operações têm a ver com a obtenção de algum tipo de valor de pagamento. Para um funcionário, o pagamento se refere aos seus ganhos. Para uma fatura, o pagamento se refere ao custo total dos bens nela listados. Podemos calcular coisas *diferentes*, como o pagamento devido para funcionários e faturas, em um *único* aplicativo de forma polimórfica? A linguagem Java oferece um recurso exigindo que classes *não relacionadas* implementem um conjunto de métodos *comuns* (por exemplo, um método que calcule um valor de pagamento)? As **interfaces** Java oferecem exatamente esse recurso.

### Padronizando interações

As interfaces definem e padronizam as maneiras pelas quais as coisas, como pessoas e sistemas, podem interagir. Por exemplo, os controles de um rádio servem como interface entre os usuários e os componentes internos do rádio. Os controles permitem que os usuários executem apenas um conjunto limitado de operações (por exemplo, mudar de estação, ajustar o volume, escolher entre AM e FM), e diferentes rádios podem implementar os controles de maneiras diferentes (por exemplo, usando botões de pressão, diais, comandos de voz). A interface especifica *quais* operações um rádio deve permitir aos usuários, mas não *como* as operações são efetuadas.

### Objetos de software se comunicam por meio de interfaces

Os objetos de software também se comunicam por meio de interfaces. Uma interface Java descreve um conjunto de métodos que podem ser chamados em um objeto para dizer a ele, por exemplo, que execute alguma tarefa ou retorne alguma informação. O próximo exemplo apresenta uma interface chamada `Payable` para descrever a funcionalidade de qualquer objeto capaz de ser pago e, assim, deve oferecer um método para determinar o valor correto do pagamento devido. A **declaração de uma interface** começa com a palavra-chave **interface** e contém apenas constantes e métodos **abstract**. Ao contrário das classes, todos os membros da interface devem ser **public**, e *as interfaces não podem especificar quaisquer detalhes de implementação*, como declarações de método concreto e variáveis de instância. Todos os métodos declarados em uma interface são implicitamente métodos **public abstract**, e todos os campos são implicitamente **public, static e final**. [Obs.: a partir do Java SE 5, tornou-se uma melhor prática

de programação declarar conjuntos de constantes, como enumerações, com a palavra-chave `enum`. Consulte a seção D.10 para ver uma introdução a `enum` e a seção F.8 para mais detalhes sobre `enum`.]



#### **Boa prática de programação G.1**

*De acordo com o Capítulo 9 da Java Language Specification, é um estilo correto declarar os métodos de uma interface sem as palavras-chave `public` e `abstract`, pois são redundantes nas declarações de método de interface. Do mesmo modo, as constantes devem ser declaradas sem as palavras-chaves `public`, `static` e `final`, pois também são redundantes.*

### **Usando uma interface**

Para usar uma interface, uma classe concreta deve especificar que implementa a interface e declarar cada método da interface com a assinatura especificada na declaração da interface. Para especificar que uma classe implementa uma interface, adicione a palavra-chave `implements` e o nome da interface ao final da primeira linha de sua declaração de classe. Uma classe que não implementa *todos* os métodos da interface é *abstrata* e deve ser declarada como `abstract`. Implementar uma interface é como assinar um *contrato* com o compilador, dizendo, “vou declarar todos os métodos especificados pela interface ou vou declarar minha classe como `abstract`”.



#### **Erro de programação comum G.6**

*Deixar de implementar qualquer método de uma interface em uma classe concreta que implementa a interface resulta em um erro de compilação indicando que a classe deve ser declarada como `abstract`.*

### **Relacionando tipos distintos**

Frequentemente, uma interface é usada quando classes distintas (isto é, não relacionadas) precisam compartilhar métodos e constantes comuns. Isso permite que objetos de classes não relacionadas sejam processados de modo polimórfico – objetos de classes que implementam a mesma interface podem responder às mesmas chamadas de método. Você pode criar uma interface descrevendo a funcionalidade desejada e, então, implementar essa interface em quaisquer classes que exijam essa funcionalidade. Por exemplo, no aplicativo de contas a pagar desenvolvido nesta seção, implementamos a interface `Payable` em qualquer classe que precise calcular um valor de pagamento (por exemplo, `Employee`, `Invoice`).

### **Interfaces versus classes abstratas**

*Frequentemente, uma interface é usada em lugar de uma classe abstrata quando não há uma implementação padrão para herdar – ou seja, nenhum campo e nenhuma implementação de método padrão. Assim como as classes `public abstract`, as interfaces normalmente são tipos `public`. Como uma classe `public`, uma interface `public` deve ser declarada em um arquivo com o mesmo nome da interface e a extensão `.java`.*

### **Interfaces de marcação (tagging)**

No Apêndice J, vamos ver a noção de “interface de marcação (tagging)” – interfaces vazias que *não* têm métodos nem valores constantes. Elas são usadas para adicionar relações *é um* às classes. Por exemplo, no Apêndice J, vamos discutir um mecanismo cha-

mado serialização de objetos, que pode converter objetos em representações de byte e converter essas representações novamente em objetos. Para permitir que esse mecanismo trabalhe com seus objetos, você simplesmente precisa marcá-los como `Serializable`, adicionando `implements Serializable` ao final da primeira linha de sua declaração de classe. Então, todos os objetos de sua classe terão uma relação *é um* com `Serializable`.

### G.12.1 Desenvolvendo uma hierarquia `Payable`

Para construir um aplicativo que possa determinar os pagamentos dos funcionários e também de faturas, primeiramente criamos a interface `Payable`, a qual contém o método `getPaymentAmount`, que retorna um valor `double` a ser pago para um objeto de qualquer classe que implemente a interface. O método `getPaymentAmount` é uma versão de uso geral do método `earnings` da hierarquia `Employee` – o método `earnings` calcula um valor de pagamento especificamente para um objeto `Employee`, enquanto `getPaymentAmount` pode ser aplicado a uma ampla gama de objetos não relacionados. Após declararmos a interface `Payable`, introduzimos a classe `Invoice`, a qual implementa a interface `Payable`. Então, modificamos a classe `Employee` de modo que também implemente a interface `Payable`. Por último, atualizamos a subclasse `SalariedEmployee` de `Employee` para “se encaixar” na hierarquia `Payable`, mudando o nome do método `earnings` de `SalariedEmployee` para `getPaymentAmount`.

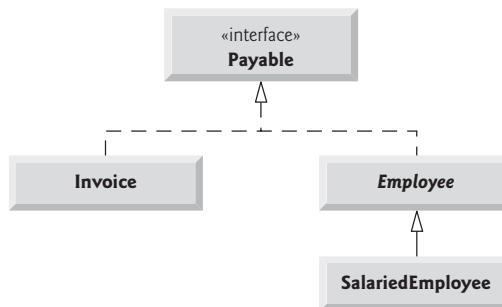


#### Boa prática de programação G.2

*Ao declarar um método em uma interface, escolha um nome que descreva a finalidade do método de maneira geral, pois ele pode ser implementado por muitas classes não relacionadas.*

As classes `Invoice` e `Employee` representam coisas para as quais a empresa precisa calcular um valor de pagamento. Ambas implementam a interface `Payable`; portanto, um programa pode chamar o método `getPaymentAmount` em objetos `Invoice` e também em objetos `Employee`. Conforme veremos em breve, isso possibilita o processamento polimórfico de objetos `Invoice` e `Employee` exigido pelo aplicativo de contas a pagar da empresa.

O diagrama de classe em UML da Fig. G.22 mostra a hierarquia usada em nosso aplicativo de contas a pagar. A hierarquia começa com a interface `Payable`. A UML diferencia uma interface das outras classes colocando a palavra “interface” entre os sinais « e » acima do nome da interface. A UML expressa a relação entre uma classe e uma interface por meio do que é conhecido como **realização**. Diz-se que uma classe “realiza” (implementa) os métodos de uma interface. Um diagrama de classe modela uma realiza-



**Figura G.22** Diagrama de classe em UML da hierarquia da interface `Payable`.

ção como uma seta tracejada, com uma cabeça vazada apontando da implementação da classe para a interface. O diagrama da Fig. G.22 indica que as classes Invoice e Employee realizam (isto é, implementam) a interface Payable. Como no diagrama de classe da Fig. G.14, a classe Employee aparece em itálico, indicando que se trata de uma classe abstrata. A classe concreta SalariedEmployee estende Employee e *herda a relação de realização de sua superclasse* com a interface Payable.

## G.12.2 Interface Payable

A declaração da interface Payable começa na linha 4 da Fig. G.23. A interface Payable contém o método `public abstract getPaymentAmount` (linha 6). O método não é declarado como `public` ou `abstract` explicitamente. Os métodos de interface são sempre `public` e `abstract`, de modo que não precisam ser declarados como tais. A interface Payable tem apenas um método – as interfaces podem ter qualquer número de métodos. Além disso, o método `getPaymentAmount` não tem parâmetros, mas os métodos de interface *podem* ter. As interfaces também podem conter campos, que são implicitamente `final` e `static`.

```

1 // Fig. G.23: Payable.java
2 // Declaração da interface Payable
3
4 public interface Payable
5 {
6 double getPaymentAmount(); // Calcula pagamento; nenhuma implementação
7 } // fim da interface Payable

```

Figura G.23 Declaração da interface Payable.

## G.12.3 Classe Invoice

Criamos agora a classe Invoice (Fig. G.24) para representar uma fatura simples que contém informações de cobrança para apenas um tipo de peça comprada. A classe declara as variáveis de instância `private partNumber`, `partDescription`, `quantity` e `pricePerItem` (nas linhas 6 a 9), que indicam o número da peça, uma descrição da peça, a quantidade pedida da peça e o preço por item. A classe Invoice contém também um construtor (linhas 12 a 19), métodos *get* e *set* (linhas 22 a 74) que manipulam as variáveis de instância da classe e um método `toString` (linhas 77 a 83) que retorna uma representação de `String` de um objeto Invoice. Os métodos `setQuantity` (linhas 46 a 52) e `setPricePerItem` (linhas 61 a 68) garantem que `quantity` e `pricePerItem` obtenham somente valores não negativos.

```

1 // Fig. G.24: Invoice.java
2 // Classe Invoice que implementa Payable.
3
4 public class Invoice implements Payable
5 {
6 private String partNumber;
7 private String partDescription;
8 private int quantity;
9 private double pricePerItem;
10
11 // construtor de quatro argumentos
12 public Invoice(String part, String description, int count,

```

Figura G.24 Classe Invoice que implementa Payable. (continua)

```
13 double price)
14 {
15 partNumber = part;
16 partDescription = description;
17 setQuantity(count); // valida e armazena a quantidade
18 setPricePerItem(price); // valida e armazena o preço por item
19 } // fim do construtor de quatro argumentos de Invoice
20
21 // configura o número da peça
22 public void setPartNumber(String part)
23 {
24 partNumber = part; // deve validar
25 } // fim do método setPartNumber
26
27 // obtém o número da peça
28 public String getPartNumber()
29 {
30 return partNumber;
31 } // fim do método getPartNumber
32
33 // configura a descrição
34 public void setPartDescription(String description)
35 {
36 partDescription = description; // deve validar
37 } // fim do método setPartDescription
38
39 // obtém a descrição
40 public String getPartDescription()
41 {
42 return partDescription;
43 } // fim do método getPartDescription
44
45 // configura a quantidade
46 public void setQuantity(int count)
47 {
48 if (count >= 0)
49 quantity = count;
50 else
51 throw new IllegalArgumentException("Quantity must be >= 0");
52 } // fim do método setQuantity
53
54 // obtém a quantidade
55 public int getQuantity()
56 {
57 return quantity;
58 } // fim do método getQuantity
59
60 // configura o preço por item
61 public void setPricePerItem(double price)
62 {
63 if (price >= 0.0)
64 pricePerItem = price;
65 else
66 throw new IllegalArgumentException(
67 "Price per item must be >= 0");
68 } // end method setPricePerItem
69
70 // obtém o preço por item
71 public double getPricePerItem()
72 {
73 return pricePerItem;
74 } // fim do método getPricePerItem
```

---

**Figura G.24** Classe Invoice que implementa Payable.

```

75
76 // retorna representação de String do objeto Invoice
77 @Override
78 public String toString()
79 {
80 return String.format("%s: \n%s: %s (%s) \n%s: %d \n%s: $%,.2f",
81 "invoice", "part number", getPartNumber(), getPartDescription(),
82 "quantity", getQuantity(), "price per item", getPricePerItem());
83 } // fim do método toString
84
85 // método exigido para colocar em prática o contrato com a interface Payable
86 @Override
87 public double getPaymentAmount()
88 {
89 return getQuantity() * getPricePerItem(); // calcula o custo total
90 } // fim do método getPaymentAmount
91 } // fim da classe Invoice

```

**Figura G.24** Classe Invoice que implementa Payable.

A linha 4 indica que a classe Invoice implementa a interface Payable. Como todas as classes, a classe Invoice também estende Object implicitamente. A linguagem Java não permite que subclasses herdem de mais de uma superclasse, mas permite que uma classe herde de uma superclasse e implemente quantas interfaces precisar. Para implementar mais de uma interface, use uma lista de nomes de interface separados por vírgulas, após a palavra-chave implements na declaração da classe, como em:

```

public class NomeDaClasse extends NomeDaSuperclasse implements
 PrimeiraInterface, SegundaInterface, ...

```



#### Observação sobre engenharia de software G.10

*Todos os objetos de uma classe que implementa várias interfaces têm uma relação é um com cada tipo de interface implementada.*

A classe Invoice implementa o único método da interface Payable – o método getPaymentAmount é declarado nas linhas 86 a 90. O método calcula o pagamento total necessário para quitar a fatura. Ele multiplica os valores de quantity e pricePerItem (obtidos por meio dos métodos get apropriados) e retorna o resultado (linha 89). Esse método satisfaz seu requisito de implementação na interface Payable – cumprimos o contrato da interface com o compilador.

### G.12.4 Modificação da classe Employee para implementar a interface Payable

Modificamos a classe Employee de modo que implemente a interface Payable. A Figura G.25 contém a classe modificada, que é idêntica à da Fig. G.16, com duas exceções. Primeiramente, a linha 4 da Fig. G.25 indica que a classe Employee agora implementa a interface Payable. Assim, precisamos mudar o nome de earnings para getPaymentAmount em toda a hierarquia Employee. Contudo, como no método earnings na versão da classe Employee da Fig. G.16, não faz sentido implementar o método getPaymentAmount na classe Employee, pois não podemos calcular o pagamento dos ganhos devidos a um objeto Employee geral – devemos primeiro saber o tipo específico de Employee. Na Fig. G.16, declaramos o método earnings como abstract por esse motivo, de modo que a classe

Employee teve de ser declarada como abstract. Isso obrigou cada subclasse concreta de Employee a sobrescrever earnings com uma implementação.

Na Fig. G.25, lidamos com essa situação de forma diferente. Lembre-se de que, quando uma classe implementa uma interface, ela faz um *contrato* com o compilador, dizendo que a classe implementará *cada um* dos métodos da interface ou que será declarada como abstract. Se for escolhida esta última opção, não precisamos declarar os métodos da interface como abstract na classe abstract – eles já são declarados implicitamente como tal na interface. Qualquer subclasse concreta da classe abstract deve implementar os métodos da interface para cumprir o contrato da superclasse com o compilador. Se a subclasse não fizer isso, também deverá ser declarada como abstract. Conforme indicado pelos comentários nas linhas 62 e 63, a classe Employee da Fig. G.25 *não* implementa o método getPaymentAmount; portanto, a classe é declarada como abstract. Cada subclasse direta de Employee *herda o contrato da superclasse* para implementar o método getPaymentAmount e, assim, deve implementar esse método para se tornar uma classe concreta, para a qual objetos podem ser instanciados. Uma classe que estende uma das subclasses concretas de Employee herdará uma implementação de getPaymentAmount e, assim, também será uma classe concreta.

```

1 // Fig. G.25: Employee.java
2 // Superclasse abstrata Employee que implementa Payable.
3
4 public abstract class Employee implements Payable
5 {
6 private String firstName;
7 private String lastName;
8 private String socialSecurityNumber;
9
10 // construtor de três argumentos
11 public Employee(String first, String last, String ssn)
12 {
13 firstName = first;
14 lastName = last;
15 socialSecurityNumber = ssn;
16 } // fim do construtor de três argumentos de Employee
17
18 // configura o nome
19 public void setFirstName(String first)
20 {
21 firstName = first; // deve validar
22 } // fim do método setFirstName
23
24 // retorna o nome
25 public String getFirstName()
26 {
27 return firstName;
28 } // fim do método getFirstName
29
30 // configura o sobrenome
31 public void setLastName(String last)
32 {
33 lastName = last; // deve validar
34 } // fim do método setLastName
35
36 // retorna o sobrenome
37 public String getLastName()
38 {

```

**Figura G.25** Superclasse abstrata Employee que implementa Payable.

```

39 return lastName;
40 } // fim do método getLastName
41
42 // configura o número da previdência social
43 public void setSocialSecurityNumber(String ssn)
44 {
45 socialSecurityNumber = ssn; // deve validar
46 } // fim do método setSocialSecurityNumber
47
48 // retorna o número da previdência social
49 public String getSocialSecurityNumber()
50 {
51 return socialSecurityNumber;
52 } // fim do método getSocialSecurityNumber
53
54 // retorna representação de String do objeto Employee
55 @Override
56 public String toString()
57 {
58 return String.format("%s %s\nsocial security number: %s",
59 getFirstName(), getLastName(), getSocialSecurityNumber());
60 } // fim do método toString
61
62 // Obs.: não implementamos o método getPaymentAmount de Payable aqui, de modo
63 // que essa classe deve ser declarada como abstract para evitar um erro de compilação.
64 } // fim da classe abstrata Employee

```

**Figura G.25** Superclasse abstrata `Employee` que implementa `Payable`.

### G.12.5 Modificação da classe `SalariedEmployee` para uso na hierarquia `Payable`

A Figura G.26 contém uma classe `SalariedEmployee` modificada que estende `Employee` e cumpre o contrato da superclasse `Employee` para implementar o método `getPaymentAmount` de `Payable`. Esta versão de `SalariedEmployee` é idêntica à da Fig. G.17, mas substituí o método `earnings` pelo método `getPaymentAmount` (linhas 34 a 38). Lembre-se de que a versão de `Payable` do método tem um nome mais *geral* para ser aplicável a classes possivelmente *distintas*. As subclasses restantes de `Employee` (por exemplo, `HourlyEmployee`, `CommissionEmployee` e `BasePlusCommissionEmployee`) também precisam ser modificadas para conter o método `getPaymentAmount` em lugar de `earnings`, a fim de refletir o fato de que agora `Employee` implementa `Payable`. Deixamos essas modificações como exercício (Exercício G.16) e usamos aqui apenas `SalariedEmployee` em nosso programa de teste. O Exercício G.17 pede para que você implemente a interface `Payable` em toda a hierarquia de classes `Employee` das Figs. G.16 a G.21 sem modificar as subclasses de `Employee`.

Quando uma classe implementa uma interface, a mesma relação *é um* fornecida pela herança se aplica. A classe `Employee` implementa `Payable`; portanto, podemos dizer que um `Employee` *é um* `Payable`. Na verdade, objetos de todas as classes que estendem `Employee` também são objetos `Payable`. Os objetos `SalariedEmployee`, por exemplo, são objetos `Payable`. Objetos de todas as subclasses da classe que implementa a interface também podem ser considerados objetos do tipo de interface. Portanto, assim como podemos atribuir a referência de um objeto `SalariedEmployee` a uma variável da superclasse `Employee`, podemos atribuir a referência de um objeto `SalariedEmployee` a uma variável da interface `Payable`. `Invoice` implementa `Payable`; portanto, um objeto `Invoice` também *é um* objeto `Payable`, e podemos atribuir a referência de um objeto `Invoice` a uma variável de `Payable`.



**Observação sobre engenharia de software G.11**

*Quando um parâmetro de método é declarado com um tipo de superclasse ou de interface, o método processa o objeto recebido como argumento de forma polimórfica.*

**Observação sobre engenharia de software G.12**

*Usando uma referência de superclasse, podemos chamar de forma polimórfica qualquer método declarado na superclasse e em suas superclasses (por exemplo, a classe Object).*

*Usando uma referência de interface, podemos chamar de forma polimórfica qualquer método declarado na interface, em suas superinterfaces (uma interface pode estender outra) e na classe Object – uma variável de um tipo de interface deve fazer referência a um objeto para chamar métodos, e todos os objetos têm os métodos da classe Object.*

```

1 // Fig. G.26: SalariedEmployee.java
2 // A classe SalariedEmployee estende Employee, a qual implementa Payable.
3
4 public class SalariedEmployee extends Employee
5 {
6 private double weeklySalary;
7
8 // construtor de quatro argumentos
9 public SalariedEmployee(String first, String last, String ssn,
10 double salary)
11 {
12 super(first, last, ssn); // passa para o construtor de Employee
13 setWeeklySalary(salary); // valida e armazena o salário
14 } // fim do construtor de quatro argumentos de SalariedEmployee
15
16 // configura o salário
17 public void setWeeklySalary(double salary)
18 {
19 if (salary >= 0.0)
20 baseSalary = salary;
21 else
22 throw new IllegalArgumentException(
23 "Weekly salary must be >= 0.0");
24 } // fim do método setWeeklySalary
25
26 // retorna o salário
27 public double getWeeklySalary()
28 {
29 return weeklySalary;
30 } // fim do método getWeeklySalary
31
32 // calcula os ganhos; implementa o método da interface Payable
33 // que era abstrato na superclasse Employee
34 @Override
35 public double getPaymentAmount()
36 {
37 return getWeeklySalary();
38 } // fim do método getPaymentAmount
39
40 // retorna representação de String do objeto SalariedEmployee
41 @Override

```

**Figura G.26** Classe `SalariedEmployee` que implementa o método `getPaymentAmount` da interface `Payable`. (continua)

```

43 {
44 return String.format("salaried employee: %s\n%s: $%,.2f",
45 super.toString(), "weekly salary", getWeeklySalary());
46 } // fim do método toString
47 } // fim da classe SalariedEmployee

```

**Figura G.26** Classe `SalariedEmployee` que implementa o método `getPaymentAmount` da interface `Payable`.

### G.12.6 Uso da interface `Payable` para processar objetos `Invoice` e `Employee` de forma polimórfica

`PayableInterfaceTest` (Fig. G.27) ilustra o fato de que a interface `Payable` pode ser usada para processar um conjunto de objetos `Invoice` e `Employee` de forma polimórfica em um único aplicativo. A linha 9 declara `payableObjects` e o atribui a um array de quatro variáveis `Payable`. As linhas 12 e 13 atribuem as referências de objetos `Invoice` aos dois primeiros elementos de `payableObjects`. Então, as linhas 14 a 17 atribuem as referências de objetos `SalariedEmployee` aos dois elementos restantes de `payableObjects`. Essas atribuições são permitidas porque um `Invoice` *é um* `Payable`, um `SalariedEmployee` *é um* `Employee` e um `Employee` *é um* `Payable`. As linhas 23 a 29 usam a instrução `for` melhorada para processar cada objeto `Payable` de forma polimórfica em `payableObjects`, imprimindo o objeto como uma `String`, junto com o valor do pagamento devido. A linha 27 chama o método `toString` por meio de uma referência de interface `Payable`, mesmo `toString` não sendo declarado na interface `Payable` – *todas as referências (inclusive as de tipos de interface) se referem a objetos que estendem `Object` e, portanto, têm um método `toString`*. (O método `toString` também pode ser chamado *implicitamente* aqui.) A linha 28 chama o método `getPaymentAmount` de `Payable` para obter o valor do pagamento para cada objeto em `payableObjects`, independente do tipo do objeto. A saída revela que as chamadas de método nas linhas 27 e 28 chamam a implementação da classe apropriada dos métodos `toString` e `getPaymentAmount`. Por exemplo, quando `currentPayable` se refere a um objeto `Invoice` durante a primeira iteração do loop `for`, são executados os métodos `toString` e `getPaymentAmount` da classe `Invoice`.

```

1 // Fig. G.27: PayableInterfaceTest.java
2 // Testa a interface Payable.
3
4 public class PayableInterfaceTest
5 {
6 public static void main(String[] args)
7 {
8 // cria array Payable de quatro elementos
9 Payable[] payableObjects = new Payable[4];
10
11 // preenche o array com objetos que implementam Payable
12 payableObjects[0] = new Invoice("01234", "seat", 2, 375.00);
13 payableObjects[1] = new Invoice("56789", "tire", 4, 79.95);
14 payableObjects[2] =
15 new SalariedEmployee("John", "Smith", "111-11-1111", 800.00);
16 payableObjects[3] =
17 new SalariedEmployee("Lisa", "Barnes", "888-88-8888", 1200.00);
18
19 System.out.println(

```

**Figura G.27** Programa de teste da interface `Payable` processando objetos `Invoice` e `Employee` de forma polimórfica. (continua)

```

20 "Invoices and Employees processed polymorphically:\n");
21
22 // processa genericamente cada elemento do array payableObjects
23 for (Payable currentPayable : payableObjects)
24 {
25 // gera currentPayable na saída e seu valor de pagamento apropriado
26 System.out.printf("%s \n%s: $%,.2f\n\n",
27 currentPayable.toString(),
28 "payment due", currentPayable.getPaymentAmount());
29 } // fim do for
30 } // fim de main
31 } // fim da classe PayableInterfaceTest

```

Invoices and Employees processed polymorphically:

```

invoice:
part number: 01234 (seat)
quantity: 2
price per item: $375.00
payment due: $750.00

invoice:
part number: 56789 (tire)
quantity: 4
price per item: $79.95
payment due: $319.80

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
payment due: $800.00

salaried employee: Lisa Barnes
social security number: 888-88-8888
weekly salary: $1,200.00
payment due: $1,200.00

```

**Figura G.27** Programa de teste da interface Payable e processando objetos Invoice e Employee de forma polimórfica.

## G.13 Interfaces comuns da API Java

Nesta seção, resumimos várias interfaces comuns encontradas na API Java. O poder e a flexibilidade das interfaces são usados com frequência por toda a API Java. Essas interfaces são implementadas e usadas da mesma maneira que as interfaces que você cria (por exemplo, a interface Payable na seção G.12.2). As interfaces da API Java permitem que você use suas próprias classes dentro das estruturas fornecidas pela linguagem Java, como a comparação de objetos de seus próprios tipos e a criação de tarefas que podem ser executadas concomitantemente com outras tarefas no mesmo programa. A Figura G.28 resume algumas interfaces da API Java comumente utilizadas.

| Interface                                                        | Descrição                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Comparable                                                       | A linguagem Java contém vários operadores de comparação (por exemplo, <, <=, >, >=, ==, !=) que permitem comparar tipos primitivos. Contudo, esses operadores <i>não podem</i> ser usados para comparar objetos. A interface Comparable é usada para permitir a comparação de objetos entre si de uma classe que a implemente com implements. Ela é comumente usada para ordenar objetos em uma coleção, como um array. |
| Serializable                                                     | Interface usada para identificar classes cujos objetos podem ser escritos (isto é, serializados) ou lidos (isto é, desserializados) em algum tipo de armazenamento (por exemplo, arquivo em disco, campo de banco de dados) ou transmitidos por uma rede.                                                                                                                                                               |
| Runnable                                                         | Implementada por qualquer classe cujos objetos devem ser capazes de executar em paralelo, usando uma técnica chamada multithread (discutida no Apêndice J). A interface contém um único método, run, o qual descreve o comportamento de um objeto, quando executado.                                                                                                                                                    |
| Interfaces receptoras de eventos de interface gráfica do usuário | Você trabalha com interfaces gráficas do usuário diariamente. Em seu navegador web, você pode digitar o endereço de um site a visitar ou clicar em um botão para voltar a um site anterior. O navegador responde à sua interação e executa a tarefa desejada. Sua interação é conhecida como evento, e o código utilizado pelo navegador para responder a um evento é conhecido como rotina de tratamento de evento.    |
| SwingConstants                                                   | Contém um conjunto de constantes usadas em programação de interface gráfica do usuário para posicionar os elementos da interface na tela.                                                                                                                                                                                                                                                                               |

**Figura G.28** Interfaces comuns da API Java.

## G.14 Para finalizar

Apresentamos a herança – a capacidade de criar classes absorvendo os membros de uma classe já existente e complementando-os com novos recursos. Você conheceu os fundamentos das superclasses e subclasses e utilizou a palavra-chave `extends` para criar uma subclasse que herdava membros de uma superclasse. Mostramos como usar a anotação `@Override` para impedir a sobrecarga involuntária, indicando que um método sobrescreve um método de superclasse. Apresentamos o modificador de acesso `protected`; os métodos de subclasse podem acessar membros `protected` da superclasse diretamente. Você aprendeu a usar `super` para acessar membros sobrescritos da superclasse. Viu também como os construtores são usados em hierarquias de herança. Em seguida, você aprendeu sobre os métodos da classe `Object`, a superclasse direta ou indireta de todas as classes Java.

Discutimos o polimorfismo – a capacidade de processar objetos que compartilham a mesma superclasse de uma hierarquia de classes, como se todos fossem objetos da superclasse. Consideramos como o polimorfismo torna os sistemas extensíveis e fáceis de manter e, então, demonstramos como usar métodos sobrescritos para realizar comportamento polimórfico. Apresentamos as classes abstratas, as quais permitem fornecer uma superclasse apropriada a partir da qual outras classes podem herdar. Você aprendeu que uma classe abstrata pode declarar métodos abstratos, os quais toda subclasse deve implementar para se tornar uma classe concreta, e que um programa pode usar variáveis de uma classe abstrata para ativar as implementações das subclasses de métodos abstratos de forma polimórfica. Aprendeu também a determinar o tipo de um objeto no momento da execução. Discutimos os conceitos de métodos e classes `final`. Por último, discutimos

a declaração e a implementação de uma interface como outro modo de obter comportamento polimórfico. Agora você já deve estar familiarizado com classes, objetos, encapsulamento, herança, polimorfismo e interfaces – os aspectos mais básicos da programação orientada a objetos. A seguir, você vai conhecer as exceções, úteis para tratamento de erros durante a execução de um programa. O tratamento de exceções ajuda a construir programas mais robustos.

---

## Exercícios de revisão (seções G.1 a G.5)

**G.1** Preencha os espaços em branco em cada um dos seguintes enunciados:

- \_\_\_\_\_ é uma forma de reutilização de software na qual novas classes adquirem os membros de classes já existentes e complementam essas classes com novos recursos.
- Os membros \_\_\_\_\_ de uma superclasse podem ser acessados na declaração da superclasse e nas declarações de subclasse.
- Em uma relação \_\_\_\_\_, um objeto de uma subclasse também pode ser tratado como um objeto de sua superclasse.
- Em uma relação \_\_\_\_\_, um objeto de uma classe tem referências para objetos de outras classes como membros.
- Na herança simples, uma classe existe em uma relação \_\_\_\_\_ com suas subclasses.
- Os membros \_\_\_\_\_ de uma superclasse são acessíveis em qualquer lugar para onde o programa tenha uma referência a um objeto dessa superclasse ou a um objeto de uma de suas subclasses.
- Quando um objeto de uma subclasse é instanciado, o \_\_\_\_\_ de uma superclasse é chamado implícita ou explicitamente.
- Os construtores de subclasse podem chamar os construtores da superclasse por meio da palavra-chave \_\_\_\_\_.

**G.2** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se um enunciado for *falso*, explique o motivo.

- Os construtores de uma superclasse não são herdados pelas subclasses.
- Uma relação *tem um* é implementada via herança.
- Uma classe Carro tem uma relação *é um* com as classes Volante e Freios.
- Quando uma subclasse redefine um método de superclasse utilizando a mesma assinatura, diz-se que a subclasse sobrecarrega o método dessa superclasse.

## Exercícios de revisão (seções G.6 a G.13)

**G.3** Preencha os espaços em branco em cada um dos seguintes enunciados:

- Se uma classe contém pelo menos um método abstrato, ela é uma classe \_\_\_\_\_.
- As classes a partir das quais objetos podem ser instanciados são denominadas classes \_\_\_\_\_.
- O \_\_\_\_\_ envolve usar uma variável de superclasse para chamar métodos em objetos de superclasse e subclasse, permitindo a você “programar no geral”.
- Os métodos que não são de interface e que não fornecem implementações devem ser declarados com a palavra-chave \_\_\_\_\_.
- Converter uma referência armazenada em uma variável de superclasse para um tipo de subclasse é chamado \_\_\_\_\_.

- G.4** Diga se cada uma das afirmativas a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.
- a) Todos os métodos de uma classe `abstract` devem ser declarados como métodos `abstract`.
  - b) Não é permitido chamar um método exclusivo de uma subclasse por meio de uma variável de subclasse.
  - c) Se uma superclasse declara um método `abstract`, uma subclasse deve implementar esse método.
  - d) Um objeto de uma classe que implementa uma interface pode ser considerado como um objeto desse tipo de interface.

## Respostas dos exercícios de revisão (seções G.1 a G.5)

- G.1** a) Herança. b) `public` e `protected`. c) *é um* ou herança. d) *tem um* ou composição. e) hierárquica. f) `public`. g) construtor. h) `super`.
- G.2** a) Verdadeira. b) Falsa. Uma relação *tem um* é implementada via composição. Uma relação *é um* é implementada via herança. c) Falsa. Esse é um exemplo de relação *tem um*. A classe `Carro` tem uma relação *é um* com a classe `Veículo`. d) Falsa. Isso é conhecido como *sobrecarregar*, não como *sobrecarregar* – um método *sobrecarregado* tem o mesmo nome, mas uma assinatura diferente.

## Respostas dos exercícios de revisão (seções G.6 a G.13)

- G.3** a) `abstract`. b) concretas. c) polimorfismo. d) `abstract`. e) *downcasting*.
- G.4** a) Falsa. Uma classe `abstract` pode conter métodos com implementações e métodos `abstract`. b) Falsa. Não é permitido chamar um método exclusivo de uma subclasse com uma variável de superclasse. c) Falsa. Somente uma subclasse concreta deve implementar o método. d) Verdadeira.

## Exercícios (seções G.1 a G.5)

- G.5** Discuta as maneiras pelas quais a herança promove a reutilização de software, economiza tempo durante o desenvolvimento de programas e ajuda a evitar erros.
- G.6** Desenhe uma hierarquia de herança para alunos de uma universidade, semelhante à hierarquia mostrada na Fig. G.2. Use `Aluno` como superclasse da hierarquia e, então, estenda `Aluno` com as classes `AlunoDeGraduação` e `AlunoGraduado`. Continue a estender a hierarquia com a maior profundidade (isto é, o máximo de níveis) possível. Por exemplo, `Calouro`, `AlunoDoSegundoAno`, `AlunoDoPenúltimoAno` e `AlunoDoÚltimoAno` poderiam estender `AlunoDeGraduação`, e `AlunoDeDoutorado` e `AlunoDeMestrado` poderiam ser subclasses de `AlunoGraduado`. Depois de desenhar a hierarquia, discuta as relações existentes entre as classes. [Obs.: você não precisa escrever um código neste exercício.]
- G.7** Alguns programadores preferem não usar acesso `protected`, pois acreditam que isso viola o encapsulamento da superclasse. Discuta os méritos relativos no uso de acesso `protected` versus uso de acesso `private` em superclasses.
- G.8** Escreva uma hierarquia de herança para as classes `Quadrilátero`, `Trapezoide`, `Paralelogramo`, `Retângulo` e `Quadrado`. Use `Quadrilátero` como superclasse da hierarquia. Crie e utilize uma classe `Ponto` para representar os pontos em cada forma. Torne a hierarquia a mais profunda (isto é, o maior número de níveis) possível. Especifique as variáveis de instância e os métodos de cada classe. As variáveis de instância `private` de `Quadrilátero` devem ser pares de

coordenadas  $x$ - $y$  para os quatro pontos extremos do Quadrilátero. Escreva um programa que instancie objetos de suas classes e gere na saída a área de cada objeto (exceto Quadrilátero).

## Exercícios (seções G.6 a G.13)

- G.9** Como o polimorfismo permite que você programe “no geral”, em vez de programar “no específico”? Discuta as principais vantagens de programar “no geral”.
- G.10** O que são métodos abstratos? Descreva as circunstâncias nas quais um método abstrato seria apropriado.
- G.11** Como o polimorfismo promove a extensibilidade?
- G.12** Discuta quatro maneiras pelas quais é possível atribuir referências de superclasse e de subclasse a variáveis de tipos da superclasse e da subclasse.
- G.13** Compare e contraste as classes abstratas e as interfaces. Por que você usaria uma classe abstrata? Por que usaria uma interface?
- G.14** (*Modificação do sistema de folha de pagamento*) Modifique o sistema de folha de pagamento das Figs. G.16 a G.21 para incluir a variável de instância `private birthDate` na classe `Employee`. Use a classe `Date` da Fig. F.7 para representar o aniversário de um funcionário. Adicione métodos `get` à classe `Date`. Presuma que a folha de pagamento é processada uma vez por mês. Crie um array de variáveis de `Employee` para armazenar referências para os vários objetos funcionário. Em um loop, calcule a folha de pagamento para cada objeto `Employee` (de forma polimórfica) e acrescente um bônus de US\$100.00 ao valor da folha de pagamento da pessoa, se o mês corrente for aquele em que cai o aniversário do objeto `Employee`.
- G.15** (*Modificação do sistema de folha de pagamento*) Modifique o sistema de folha de pagamento das Figs. G.16 a G.21 para incluir a subclasse adicional `PieceWorker` de `Employee`, representando um funcionário cujo pagamento se baseia no número de peças de mercadorias produzidas. A classe `PieceWorker` deve conter as variáveis de instância `private wage` (para armazenar a remuneração por peça do funcionário) e `pieces` (para armazenar o número de peças produzidas). Forneça uma implementação concreta do método `earnings` na classe `PieceWorker`, para calcular os ganhos do funcionário multiplicando o número de peças produzidas pela remuneração por peça. Crie um array de variáveis de `Employee` para armazenar referências para objetos de cada classe concreta na nova hierarquia `Employee`. Para cada `Employee`, exiba sua representação de `String` e os ganhos.
- G.16** (*Modificação do sistema de contas a pagar*) Neste exercício, modificamos o aplicativo de contas a pagar das Figs. G.23 a G.27 para incluir a funcionalidade completa do aplicativo de folha de pagamento das Figs. G.16 a G.21. O aplicativo ainda deve processar dois objetos `Invoice`, mas agora deve processar um objeto de cada uma das quatro subclasses de `Employee`. Se o objeto que está sendo processado é um `BasePlusCommissionEmployee`, o aplicativo deve aumentar em 10% o salário-base do `BasePlusCommissionEmployee`. Por fim, o aplicativo deve gerar na saída o valor do pagamento de cada objeto. Complete os passos a seguir para criar o novo aplicativo:
- Modifique as classes `HourlyEmployee` (Fig. G.18) e `CommissionEmployee` (Fig. G.19) para colocá-las na hierarquia `Payable` como subclasses da versão de `Employee` (Fig. G.25) que implementa `Payable`. [Dica: mude o nome do método `earnings` para `getPaymentAmount` em cada subclasse, para que a classe satisfaça seu contrato herdado com a interface `Payable`.]
  - Modifique a classe `BasePlusCommissionEmployee` (Fig. G.20) de modo que ela estenda a versão da classe `CommissionEmployee` criada na parte (a).

- c) Modifique `PayableInterfaceTest` (Fig. G.27) para processar de forma polimórfica dois objetos `Invoices`, um `SalariedEmployee`, um `HourlyEmployee`, um `CommissionEmployee` e um `BasePlusCommissionEmployee`. Primeiramente, gere uma representação de `String` de cada objeto `Payable`. Em seguida, se um objeto é um `BasePlusCommissionEmployee`, aumente seu salário-base em 10%. Por fim, gere o valor do pagamento de cada objeto `Payable`.

**G.17 (Modificação do sistema de contas a pagar)** É possível incluir a funcionalidade do aplicativo de folha de pagamento (Figs. G.16 a G.21) no aplicativo de contas a pagar, sem modificar as subclasses `SalariedEmployee`, `HourlyEmployee`, `CommissionEmployee` ou `BasePlusCommissionEmployee` de `Employee`. Para isso, você pode modificar a classe `Employee` (Fig. G.16) para implementar a interface `Payable` e declarar o método `getPaymentAmount` para chamar o método `earnings`. Então, o método `getPaymentAmount` seria herdado pelas subclasses da hierarquia `Employee`. Quando `getPaymentAmount` fosse chamado por um objeto de subclasse em particular, chamaria o método `earnings` apropriado dessa subclasse de forma polimórfica. Reimplemente o Exercício G.16 usando a hierarquia `Employee` original do aplicativo de folha de pagamento das Figs. G.16 a G.21. Modifique a classe `Employee` conforme descrito neste exercício e *não* modifique as subclasses da classe `Employee`.



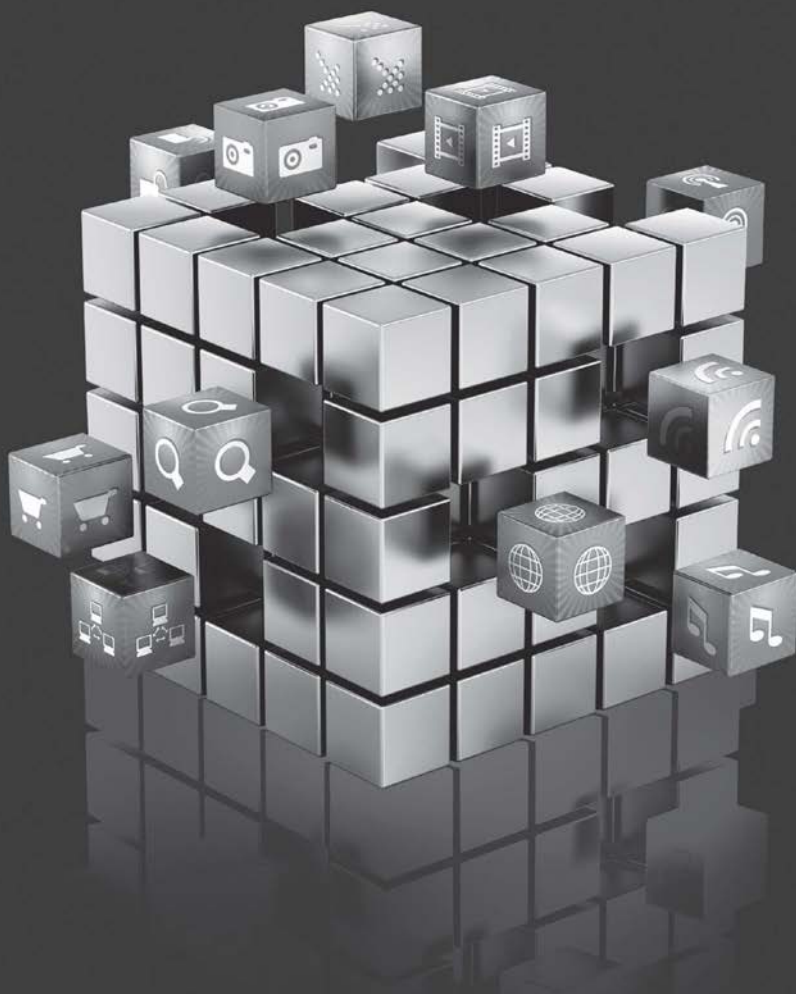
# H

## Tratamento de exceções: uma investigação mais aprofundada

### Objetivos

Neste capítulo, você vai:

- Aprender o que são exceções e como são tratadas.
- Saber quando deve usar tratamento de exceção.
- Usar blocos `try` para delimitar código no qual podem ocorrer exceções.
- Lançar (`throw`) exceções para indicar um problema.
- Usar blocos `catch` para especificar rotinas de tratamento de exceção.
- Usar o bloco `finally` para liberar recursos.
- Conhecer a hierarquia de classes de exceção.



- |                                                                                                                   |                                                                              |
|-------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| <b>H.1</b> Introdução                                                                                             | <b>H.5</b> Hierarquia de exceções da linguagem Java                          |
| <b>H.2</b> Exemplo: divisão por zero sem tratamento de exceção                                                    | <b>H.6</b> Bloco <code>finally</code>                                        |
| <b>H.3</b> Exemplo: tratamento de exceções <code>ArithmeticException</code> e <code>InputMismatchException</code> | <b>H.7</b> Stack unwinding e obtenção de informações de um objeto de exceção |
| <b>H.4</b> Quando usar tratamento de exceção                                                                      | <b>H.8</b> Para finalizar                                                    |

*Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios*

## H.1 Introdução

Uma exceção (exception) é uma indicação de um problema ocorrido durante a execução de um programa. O tratamento de exceções permite criar aplicativos que podem solucionar (ou tratar) exceções. Em muitos casos, o tratamento de uma exceção permite que um programa continue a ser executado como se não tivesse encontrado problema. Os recursos apresentados neste apêndice o ajudam a escrever programas robustos que podem lidar com problemas e continuar a ser executados ou terminar normalmente.

## H.2 Exemplo: divisão por zero sem tratamento de exceção

Primeiramente, demonstramos o que acontece quando surgem erros em um aplicativo que não usa tratamento de exceção. A Figura H.1 solicita do usuário dois valores inteiros e os passa para o método `quotient`, o qual calcula o quociente inteiro e retorna um resultado `int`. Nesse exemplo, você vai ver que as exceções são **lançadas** (isto é, a exceção ocorre) quando um método detecta um problema e é incapaz de resolvê-lo.

```

1 // Fig. H.1: DivideByZeroNoExceptionHandling.java
2 // Divisão de valor inteiro sem tratamento de exceção.
3 import java.util.Scanner;
4
5 public class DivideByZeroNoExceptionHandling
6 {
7 // demonstra o lançamento de uma exceção quando ocorre uma divisão por zero
8 public static int quotient(int numerator, int denominator)
9 {
10 return numerator / denominator; // possível divisão por zero
11 } // fim do método quotient
12
13 public static void main(String[] args)
14 {
15 Scanner scanner = new Scanner(System.in); // scanner para entrada
16
17 System.out.print("Please enter an integer numerator: ");
18 int numerator = scanner.nextInt();
19 System.out.print("Please enter an integer denominator: ");
20 int denominator = scanner.nextInt();
21
22 int result = quotient(numerator, denominator);
23 System.out.printf(

```

**Figura H.1** Divisão de valor inteiro sem tratamento de exceção. (continua)

```

24 "\nResult: %d / %d = %d\n", numerator, denominator, result);
25 } // fim de main
26 } // fim da classe DivideByZeroNoExceptionHandling

```

```

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14

```

```

Please enter an integer numerator: 100
Please enter an integer denominator: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
 at DivideByZeroNoExceptionHandling.quotient(
 DivideByZeroNoExceptionHandling.java:10)
 at DivideByZeroNoExceptionHandling.main(
 DivideByZeroNoExceptionHandling.java:22)

```

```

Please enter an integer numerator: 100
Please enter an integer denominator: hello
Exception in thread "main" java.util.InputMismatchException
 at java.util.Scanner.throwFor(Unknown Source)
 at java.util.Scanner.next(Unknown Source)
 at java.util.Scanner.nextInt(Unknown Source)
 at java.util.Scanner.nextInt(Unknown Source)
 at DivideByZeroNoExceptionHandling.main(
 DivideByZeroNoExceptionHandling.java:20)

```

**Figura H.1** Divisão de valor inteiro sem tratamento de exceção.

O primeiro exemplo de execução na Fig. H.1 mostra uma divisão bem-sucedida. Na segunda execução, o usuário digita o valor 0 como denominador. Várias linhas de informação são exibidas em resposta a essa entrada inválida. Essas informações são conhecidas como **rastreamento de pilha**, o que inclui o nome da exceção (`java.lang.ArithmeticException`) em uma mensagem descritiva indicando o problema ocorrido e a pilha de chamada de métodos (isto é, o encadeamento de chamadas) no momento em que ela ocorreu. O rastreamento de pilha inclui o caminho de execução, método por método, que levou à exceção. Isso ajuda a depurar o programa. A primeira linha específica que ocorreu uma exceção `ArithmeticException`. O texto após o nome da exceção (“/ by zero”) indica que ela ocorreu como resultado de uma tentativa de dividir por zero. A linguagem Java não permite divisão por zero na aritmética de valores inteiros. Quando isso ocorre, ela lança uma exceção `ArithmeticException`. As exceções `ArithmeticException` podem surgir a partir de vários problemas diferentes na aritmética; portanto, os dados extras (“/ by zero”) fornecem informações mais específicas. A linguagem Java *permite* divisão por zero com valores de ponto flutuante. Tal cálculo resulta no valor infinito positivo ou negativo, o qual é representado em Java como um valor de ponto flutuante (mas exibido como a string `Infinity` ou `-Infinity`). Se 0.0 é dividido por 0.0, o resultado é NaN (not a number – não é um número), o qual também é representado em Java como um valor de ponto flutuante (mas aparece como NaN).

A partir da última linha do rastreamento de pilha, vemos que a exceção foi detectada na linha 22 do método `main`. Cada linha do rastreamento de pilha contém o nome da classe e o método (`DivideByZeroNoExceptionHandling.main`), seguidos pelo nome do arquivo e do número da linha (`DivideByZeroNoExceptionHandling.java:22`). Subindo no rastreamento de pilha, vemos que a exceção ocorre na linha 10 do método `quotient`. A linha superior do encadeamento de chamadas indica o **ponto de lançamento** – o ponto inicial em que a exceção ocorreu. O ponto de lançamento dessa exceção está na linha 10 do método `quotient`.

Na terceira execução, o usuário digita a string "hello" como denominador. Observe, novamente, que aparece um rastreamento de pilha. Isso nos informa que ocorreu uma exceção `InputMismatchException` (pacote `java.util`). Nossos exemplos anteriores que liam valores numéricos do usuário presumiam que ele digitava um valor inteiro correto. Contudo, às vezes os usuários cometem erros e digitam valores não inteiros. Quando o método `nextInt` de `Scanner` recebe uma string que não representa um valor inteiro válido, ocorre uma exceção `InputMismatchException`. A partir do fim do rastreamento de pilha, vemos que a exceção foi detectada na linha 20 do método `main`. Subindo no rastreamento de pilha, vemos que a exceção ocorreu no método `nextInt`. Observe que, no lugar do nome de arquivo e do número da linha, recebemos o texto `Unknown Source`. Isso significa que os assim chamados símbolos de depuração, que fornecem as informações de nome de arquivo e o número de linha da classe desse método, não estavam disponíveis para a JVM – isso acontece normalmente para as classes da API Java. Muitos IDEs têm acesso ao código-fonte da API Java e exibirão nomes de arquivo e número de linha nos rastreamentos de pilha.

Nos exemplos de execução da Fig. H.1, quando ocorrem exceções e os rastreamentos são exibidos, o programa também é encerrado. Isso nem sempre ocorre no Java – às vezes, um programa pode continuar, mesmo tendo ocorrido uma exceção e tendo sido impresso um rastreamento de pilha. Nesses casos, o aplicativo poderá produzir resultados inesperados. Por exemplo, um aplicativo com interface gráfica do usuário frequentemente continuará a ser executado. A próxima seção demonstra como tratar essas exceções.

Na Fig. H.1, os dois tipos de exceções foram detectados no método `main`. No próximo exemplo, vamos ver como tratar essas exceções para permitir que o programa seja executado até o término normal.

### H.3 Exemplo: tratamento de exceções `ArithmeticException` e `InputMismatchException`

O aplicativo da Fig. H.2, que é baseado na Fig. H.1, utiliza tratamento de exceção para processar quaisquer exceções `ArithmeticException` e `InputMismatchException` que surjam. O aplicativo ainda solicita do usuário dois valores inteiros e os passa para o método `quotient`, o qual calcula o quociente inteiro e retorna um resultado `int`. Esta versão do aplicativo utiliza tratamento de exceção para que, se o usuário cometer um erro, o programa capture e trate (isto é, se encarregue) a exceção – neste caso, permitindo ao usuário digitar a entrada novamente.

```

1 // Fig. H.2: DivideByZeroWithExceptionHandling.java
2 // Tratamento de exceções ArithmeticException e InputMismatchException.
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6 public class DivideByZeroWithExceptionHandling
7 {
8 // demonstra o lançamento de uma exceção quando ocorre uma divisão por zero
9 public static int quotient(int numerator, int denominator)
10 throws ArithmeticException
11 {
12 return numerator / denominator; // possível divisão por zero
13 } // fim do método quotient
14
15 public static void main(String[] args)
16 {
17 Scanner scanner = new Scanner(System.in); // scanner para entrada
18 boolean continueLoop = true; // determina se mais entrada é necessária
19
20 do
21 {
22 try // lê dois números e calcula o quociente
23 {
24 System.out.print("Please enter an integer numerator: ");
25 int numerator = scanner.nextInt();
26 System.out.print("Please enter an integer denominator: ");
27 int denominator = scanner.nextInt();
28
29 int result = quotient(numerator, denominator);
30 System.out.printf("\nResult: %d / %d = %d\n", numerator,
31 denominator, result);
32 continueLoop = false; // entrada bem-sucedida; fim do loop
33 } // fim do try
34 catch (InputMismatchException inputMismatchException)
35 {
36 System.err.printf("\nException: %s\n",
37 inputMismatchException);
38 scanner.nextLine(); // descarta entrada para o usuário tentar novamente
39 System.out.println(
40 "You must enter integers. Please try again.\n");
41 } // fim do catch
42 catch (ArithmeticException arithmeticException)
43 {
44 System.err.printf("\nException: %s\n", arithmeticException);
45 System.out.println(
46 "Zero is an invalid denominator. Please try again.\n");
47 } // fim do catch
48 } while (continueLoop); // fim do do...while
49 } // fim de main
50 } // fim da classe DivideByZeroWithExceptionHandling

```

```

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14

```

**Figura H.2** Tratamento de exceções ArithmeticException e InputMismatchException. (continua)

```
Please enter an integer numerator: 100
Please enter an integer denominator: 0

Exception: java.lang.ArithmeticException: / by zero
Zero is an invalid denominator. Please try again.

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: hello

Exception: java.util.InputMismatchException
You must enter integers. Please try again.

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

**Figura H.2** Tratamento de exceções `ArithmeticException` e `InputMismatchException`.

O primeiro exemplo de execução na Fig. H.2 é bem-sucedido e não encontra nenhum problema. Na segunda execução, o usuário digita um denominador zero e ocorre uma exceção `ArithmeticException`. Na terceira execução, o usuário digita a string "hello" como denominador e ocorre uma exceção `InputMismatchException`. Para cada exceção, o usuário é informado do erro e instado a tentar outra vez, digitando dois novos valores inteiros. Em cada exemplo de execução, o programa funciona com êxito até a conclusão. A classe `InputMismatchException` é importada na linha 3. A classe `ArithmeticException` não precisa ser importada, pois está no pacote `java.lang`. A linha 18 cria a variável boolean `continueLoop`, a qual é `true` se o usuário ainda não digitou uma entrada válida. As linhas 20 a 48 solicitam repetidamente para que o usuário digite uma entrada até que um valor válido seja recebido.

### **Englobando código em um bloco `try`**

As linhas 22 a 33 contêm um **bloco `try`**, o qual engloba o código que pode lançar uma exceção e o código que não deve ser executado se ocorrer uma exceção (isto é, se uma exceção ocorrer, o código restante do bloco `try` será pulado). Um bloco `try` consiste na palavra-chave `try`, seguida por um bloco de código incluído entre chaves. [*Obs.*: às vezes, o termo “bloco `try`” se refere apenas ao bloco de código que vem após a palavra-chave `try` (não incluindo a palavra-chave `try` em si). Por simplicidade, usamos o termo “bloco `try`” para nos referirmos ao bloco de código que vem após a palavra-chave `try`, incluindo a palavra-chave.] As instruções que leem os valores inteiros do teclado (linhas 25 e 27) usam cada uma o método `nextInt` para ler um `int`. O método `nextInt` lança uma exceção `InputMismatchException` se o valor lido não é inteiro.

A divisão que pode causar uma exceção `ArithmeticException` não é executada no bloco `try`. Em vez disso, a chamada ao método `quotient` (linha 29) ativa o código que tenta a divisão (linha 12); a JVM lança um objeto `ArithmeticException` quando o denominador é zero.



### Observação sobre engenharia de software H.1

*Exceções podem surgir por meio de código explicitamente mencionado em um bloco try, por meio de chamadas a outros métodos, por meio de chamadas de método profundamente aninhadas, iniciadas por código de um bloco try ou a partir da Java Virtual Machine, quando ela executa códigos binários (bytecodes) Java.*

### Capturando exceções

Neste exemplo, o bloco try é seguido por dois blocos catch – um que trata uma exceção `InputMismatchException` (linhas 34 a 41) e outro que trata uma exceção `ArithmeticException` (linhas 42 a 47). Um **bloco catch** (também chamado de **cláusula catch** ou **rotina de tratamento de exceção**) captura (isto é, recebe) e trata uma exceção. Um bloco catch começa com a palavra-chave `catch` e é seguido por um parâmetro entre parênteses (chamado de parâmetro de exceção, a ser discutido em breve) e por um bloco de código colocado entre chaves. [Obs.: às vezes, o termo “cláusula catch” é usado para se referir à palavra-chave `catch`, seguida por um bloco de código, enquanto o termo “bloco catch” se refere apenas ao bloco de código que vem depois da palavra-chave `catch`, mas não a inclui. Por simplicidade, usamos o termo “bloco catch” para nos referirmos ao bloco de código que vem após a palavra-chave `catch`, incluindo a própria palavra-chave.]

Pelo menos um bloco catch ou um **bloco finally** (discutido na seção H.6) deve vir imediatamente após o bloco try. Cada bloco catch especifica nos parênteses um **parâmetro de exceção** identificando o tipo de exceção que a rotina de tratamento pode processar. Quando ocorre uma exceção em um bloco try, o bloco catch executado é o primeiro cujo tipo corresponde ao tipo da exceção ocorrida (isto é, o tipo no bloco catch corresponde exatamente ao tipo da exceção lançada ou é uma superclasse dele). O nome do parâmetro de exceção permite ao bloco catch interagir com um objeto de exceção capturado – por exemplo, chamar implicitamente o método `toString` da exceção capturada (como nas linhas 37 e 44), o que exibe informações básicas sobre a exceção. Observe que usamos o **objeto `System.err` (fluxo de erro padrão)** para gerar mensagens de erro na saída. Por padrão, os métodos `print` de `System.err`, assim como os de `System.out`, exibem dados no prompt de comando.

A linha 38 do primeiro bloco catch chama o método `nextLine` de `Scanner`. Como ocorreu uma exceção `InputMismatchException`, a chamada ao método `nextInt` nunca lê os dados do usuário – assim, lemos essa entrada com uma chamada ao método `nextLine`. Não fazemos nada com a entrada neste ponto, pois sabemos que é inválida. Cada bloco catch exibe uma mensagem de erro e pede para o usuário tentar novamente. Depois que um ou outro bloco catch termina, é solicitada a entrada ao usuário. Em breve, vamos examinar mais detidamente como esse fluxo de controle funciona no tratamento de exceções.



### Erro de programação comum H.1

*É um erro de sintaxe colocar código entre um bloco try e seus blocos catch correspondentes.*



### Erro de programação comum H.2

*Cada bloco catch pode ter apenas um parâmetro – especificar uma lista de parâmetros de exceção separados por vírgulas é um erro de sintaxe.*

Uma **exceção não capturada** é aquela para a qual não existe um bloco `catch` correspondente. Você viu exceções não capturadas na segunda e terceira saídas da Fig. H.1. Lembre-se de que, quando ocorriam exceções naquele exemplo, o aplicativo terminava prematuramente (depois de exibir o rastreamento de pilha da exceção). Isso nem sempre ocorre como resultado de exceções não capturadas. A linguagem Java utiliza um modelo “multithread” de execução de programa – cada **thread** é uma atividade paralela. Um programa pode ter muitas threads. Se um programa tem apenas uma thread, uma exceção não capturada o fará terminar. Se um programa tem várias threads, uma exceção não capturada terminará *apenas* a thread onde a exceção ocorreu. Contudo, nesses programas, certas threads podem depender de outras e, se uma thread terminar devido a uma exceção não capturada, poderá haver efeitos adversos no resto do programa. O Apêndice J discute essas questões.

### Modelo de terminação do tratamento de exceções

Se ocorre uma exceção em um bloco `try` (como `InputMismatchException` sendo lançada como resultado do código na linha 25 da Fig. H.2), o bloco `try` termina imediatamente e o controle do programa é transferido para o *primeiro* dos blocos `catch` seguintes, no qual o tipo de parâmetro de exceção corresponde ao tipo da exceção lançada. Na Fig. H.2, o primeiro bloco `catch` captura exceções `InputMismatchException` (que ocorrem se é digitada uma entrada inválida), e o segundo bloco `catch` captura exceções `ArithmeticException` (que ocorrem se é feita uma tentativa de dividir por zero). Após a exceção ser tratada, o controle do programa *não* retorna para o ponto de lançamento, pois o bloco `try` *expirou* (e suas variáveis locais foram perdidas). Em vez disso, o controle é retomado após o último bloco `catch`. Isso é conhecido como **modelo de terminação de tratamento de exceção**. Algumas linguagens usam o **modelo de retomada de tratamento de exceção**, no qual, após a exceção ser tratada, o controle é retomado imediatamente para depois do ponto de lançamento. Observe que demos nomes aos nossos parâmetros de exceção (`inputMismatchException` e `arithmeticException`) de acordo com seus tipos. Os programadores Java frequentemente usam apenas a letra e como nome de seus parâmetros de exceção.

Após a execução de um bloco `catch`, o fluxo de controle desse programa passa para a primeira instrução após o último bloco `catch` (linha 48, neste caso). A condição na instrução `do...while` é `true` (a variável `continueLoop` contém seu valor `true` inicial); portanto, o controle retorna para o início do loop e o usuário é mais uma vez instado a fazer uma entrada. Essa instrução de controle fará loop até que uma entrada válida seja inserida. Nesse ponto, o controle do programa chega à linha 32, a qual atribui `false` à variável `continueLoop`. Então, o bloco `try` termina. Se não é lançada uma exceção no bloco `try`, os blocos `catch` são pulados e o controle continua com a primeira instrução após os blocos `catch` (vamos aprender sobre outra possibilidade quando discutirmos o bloco `finally`, na seção H.6). Agora a condição do loop `do...while` é `false` e o método `main` termina.

O bloco `try` e seus blocos `catch` e/ou `finally` correspondentes formam uma **instrução try**. Não confunda os termos “bloco `try`” e “instrução `try`” – este último inclui o bloco `try` e os blocos `catch` e/ou `finally` seguintes.

Como qualquer outro bloco de código, quando um bloco `try` termina, as variáveis locais nele declaradas saem do escopo e não são mais acessíveis; assim, as variáveis locais de um bloco `try` não são acessíveis nos blocos `catch` correspondentes. Quando um bloco `catch` termina, as variáveis locais declaradas dentro dele (incluindo o parâmetro de exceção desse bloco `catch`) também saem do escopo e são destruídas. Quaisquer blocos `catch`



restantes na instrução `try` são ignorados, e a execução é retomada na primeira linha de código após a sequência `try...catch` – será um bloco `finally`, se houver um.

### Usando a cláusula `throws`

Agora, vamos examinar o método `quotient` (Fig. H.2, linhas 9 a 13). A parte da declaração do método, localizada na linha 10, é conhecida como **cláusula `throws`**. Ela especifica as exceções lançadas pelo método. Essa cláusula aparece *depois* da lista de parâmetros do método e *antes* do corpo do método. Ela contém uma lista separada por vírgulas das exceções que o método vai lançar caso vários tipos de problemas ocorram. Essas exceções podem ser lançadas por instruções do corpo do método ou por métodos chamados a partir do corpo. Um método pode lançar exceções das classes listadas em sua cláusula `throws` ou de suas subclasses. Adicionamos a cláusula `throws` a esse aplicativo para indicar ao resto do programa que esse método pode lançar uma exceção `ArithmeticException`. Assim, os clientes do método `quotient` são informados de que ele pode lançar uma exceção `ArithmeticException`. Você vai aprender mais sobre a cláusula `throws` na seção H.5.

Quando a linha 12 é executada, se `denominator` é zero, a JVM lança um objeto `ArithmeticException`. Esse objeto será capturado pelo bloco `catch` nas linhas 42 a 47, o qual exibe informações básicas sobre a exceção chamando implicitamente o método `toString` da exceção e, então, pede ao usuário para que tente de novo.

Se `denominator` não é zero, o método `quotient` faz a divisão e retorna o resultado no ponto de sua chamada no bloco `try` (linha 29). As linhas 30 e 31 exibem o resultado do cálculo e a linha 32 configura `continueLoop` como `false`. Nesse caso, o bloco `try` termina com êxito, de modo que o programa pula os blocos `catch` e não executa a condição da linha 48; assim, o método `main` completa sua execução normalmente.

Quando `quotient` lança uma exceção `ArithmeticException`, `quotient` termina e não retorna um valor; as variáveis locais de `quotient` saem do escopo (e são destruídas). Se `quotient` contivesse variáveis locais que fossem referências para objetos e não houvesse outras referências para eles, os objetos seriam marcados para a coleta de lixo. Além disso, quando ocorre uma exceção, o bloco `try` a partir do qual `quotient` foi chamado termina antes que as linhas 30 a 32 possam ser executadas. Aqui, também, se fossem criadas variáveis locais no bloco `try`, antes da exceção ser lançada, elas sairiam do escopo.

Se uma exceção `InputMismatchException` é gerada pelas linhas 25 ou 27, o bloco `try` termina e a execução continua com o bloco `catch` nas linhas 34 a 41. Nesse caso, o método `quotient` não é chamado. Então, o método `main` continua após o último bloco (linha 48).

## H.4 Quando usar tratamento de exceção

O tratamento de exceção é projetado para processar **erros síncronos**, os quais ocorrem quando uma instrução é executada. Exemplos comuns que vamos ver por todo o livro são os índices de array fora do intervalo, estouro aritmético (isto é, um valor fora do intervalo de valores representável), divisão por zero, parâmetros de método inválidos, interrupção de thread (conforme veremos no Apêndice J) e alocação de memória malsucedida (devida à falta de memória). O tratamento de exceções não é feito para processar problemas associados a **eventos assíncronos** (por exemplo, conclusões de E/S de disco, chegadas de mensagem de rede, cliques de mouse e toques de tecla), os quais ocorrem em paralelo e são independentes do fluxo de controle do programa.

## H.5 Hierarquia de exceções da linguagem Java

Todas as classes de exceção Java herdam direta ou indiretamente da classe **Exception**, formando uma hierarquia de herança. Você pode ampliar essa hierarquia com suas próprias classes de exceção. A classe **Throwable** (uma subclasse de **Object**) é a superclasse da classe **Exception**. Somente objetos **Throwable** podem ser usados com o mecanismo de tratamento de exceção. A classe **Throwable** tem duas subclasses: **Exception** e **Error**. A classe **Exception** e suas subclasses – por exemplo, **RuntimeException** (pacote `java.lang`) e **IOException** (pacote `java.io`) – representam situações excepcionais que podem ocorrer em um programa Java e que podem ser capturadas pelo aplicativo. A classe **Error** e suas subclasses representam situações anormais que acontecem na JVM. A maioria das situações de *Error acontece raramente e não deve ser capturada pelos aplicativos – normalmente não é possível os aplicativos se recuperarem de situações de Error.*

### Exceções verificadas versus não verificadas

A linguagem Java distingue entre **exceções verificadas** e **exceções não verificadas**. Essa distinção é importante, pois o compilador Java impõe o **requisito de capturar ou declarar** (`catch-or-declare`) para exceções verificadas. O tipo de uma exceção determina se ela é verificada ou não. Todos os tipos de exceção que são subclasses diretas ou indiretas da classe **RuntimeException** (pacote `java.lang`) são exceções não verificadas. Normalmente, elas são causadas por defeitos no código de seu programa. Exemplos de exceções não verificadas incluem **ArrayIndexOutOfBoundsException** (discutida no Apêndice E) e **ArithmeticException**. Todas as classes que herdam da classe **Exception**, mas não da classe **RuntimeException**, são consideradas exceções verificadas. Tais exceções normalmente são causadas por condições que não estão sob o controle do programa – por exemplo, no processamento de arquivos, o programa não consegue abrir um arquivo porque ele não existe. As classes que herdam da classe **Error** são consideradas não verificadas.

O compilador *verifica* cada chamada e cada declaração de método para determinar se o método lança exceções verificadas. Em caso positivo, o compilador verifica se a exceção verificada é capturada ou está declarada em uma cláusula `throws`. Mostramos como capturar e declarar exceções verificadas nos próximos exemplos. Lembre-se, da seção H.3, de que a cláusula `throws` especifica as exceções lançadas por um método. Tais exceções não são capturadas no corpo do método. Para satisfazer a parte *capturar* do requisito de capturar ou declarar, o código que gera a exceção deve ser envolto em um bloco `try` e fornecer uma rotina de tratamento `catch` para o tipo de exceção verificada (ou um de seus tipos de superclasse). Para satisfazer a parte *declarar* do requisito, o método que contém o código que gera a exceção deve fornecer uma cláusula `throws` contendo o tipo de exceção verificada, depois de sua lista de parâmetros e antes do seu corpo. Se o requisito de capturar ou declarar não for satisfeito, o compilador gerará uma mensagem de erro indicando que a exceção deve ser capturada ou declarada. Isso o obriga a pensar sobre os problemas que podem ocorrer quando um método que lança exceções verificadas é chamado.



### Observação sobre engenharia de software H.2

*Você deve trabalhar com exceções verificadas. Isso resulta em código mais robusto do que seria criado se as exceções fossem simplesmente ignoradas.*



### Erro de programação comum H.3

*Ocorre um erro de compilação se um método tenta lançar uma exceção verificada explicitamente (ou chama outro método que lança uma exceção verificada) e essa exceção não está listada na sua cláusula throws.*



### Erro de programação comum H.4

*Se um método de subclasse sobrescreve um método de superclasse, é um erro o método de subclasse listar mais exceções em sua cláusula throws que o método de superclasse sobrescreve. Contudo, a cláusula throws de uma subclasse pode conter um subconjunto da lista de throws de uma superclasse.*



### Observação sobre engenharia de software H.3

*Se seu método chama outros métodos que lançam exceções verificadas, essas exceções devem ser capturadas ou declaradas no seu método. Se uma exceção pode ser tratada de forma significativa em um método, o método deve capturá-la, em vez de declará-la.*

Ao contrário das exceções verificadas, o compilador Java *não* verifica o código para determinar se uma exceção não verificada é capturada ou declarada. Normalmente, as exceções não verificadas podem ser evitadas por uma codificação correta. Por exemplo, a exceção não verificada `ArithmeticException`, lançada pelo método `quotient` (linhas 9 a 13) na Fig. H.2, pode ser evitada se o método garantir que o denominador não seja zero *antes* de tentar efetuar a divisão. As exceções não verificadas não precisam ser listadas na cláusula `throws` de um método – mesmo que sejam, não é exigido que tais exceções sejam capturadas por um aplicativo.



### Observação sobre engenharia de software H.4

*Embora o compilador não imponha o requisito de capturar ou declarar para exceções não verificadas, forneça código de tratamento de exceção apropriado quando souber que tais exceções podem ocorrer. Por exemplo, um programa deve processar a exceção `NumberFormatException` do método `Integer.parseInt`, mesmo `NumberFormatException` (uma subclasse indireta de `RuntimeException`) sendo um tipo de exceção não verificada. Isso torna seus programas mais robustos.*

## Capturando exceções de subclasse

Se uma rotina de tratamento `catch` é escrita para capturar objetos de exceção do tipo da superclasse, ela também pode capturar todos os objetos das subclasses dessa classe. Isso permite que `catch` trate erros relacionados com uma notação concisa e possibilita o processamento polimórfico de exceções relacionadas. Certamente você pode capturar cada tipo de subclasse individualmente, caso essas exceções exijam processamentos diferentes.

## Somente o primeiro catch correspondente é executado

Se existem *vários* blocos `catch` que correspondem a um tipo de exceção em particular, somente o *primeiro* bloco `catch` correspondente é executado quando ocorre uma exceção desse tipo. Causa um erro de compilação capturar *exatamente o mesmo tipo* em dois blocos `catch` diferentes, associados a um bloco `try` em particular. No entanto, pode haver vários blocos `catch` que correspondam a uma exceção – isto é, vários blocos `catch` cujos tipos sejam iguais ao tipo de exceção ou a uma superclasse desse tipo. Por exemplo, depois de um bloco `catch` para o tipo `ArithmeticException`, poderíamos colocar um bloco

catch para o tipo Exception – ambos corresponderiam a exceções ArithmeticException, mas somente o primeiro bloco catch correspondente seria executado.



#### Dica de prevenção de erro H.1

*A captura de tipos de subclasse individualmente ficará sujeita a erros, caso você se esqueça de testar um ou mais dos tipos de subclasse explicitamente; capturar a superclasse garante que os objetos de todas as subclasses sejam capturados. Colocar um bloco catch para o tipo de superclasse depois de todos os outros blocos catch de subclasse garante que todas as exceções de subclasse sejam finalmente capturadas.*



#### Erro de programação comum H.5

*Colocar um bloco catch para um tipo de exceção de superclasse antes de outros blocos catch que capturam tipos de exceção de subclasse impediria a execução desses blocos catch; portanto, ocorreria um erro de compilação.*

## H.6 Bloco finally

Os programas que obtêm certos tipos de recursos devem retorná-los explicitamente para o sistema a fim de evitar os assim chamados **vazamentos de recursos**. Em linguagens de programação como C e C++, o tipo mais comum de vazamento de recursos é o vazamento de memória. A linguagem Java faz coleta de lixo automática para memória não mais utilizada pelos programas, evitando assim a maioria dos vazamentos de memória. Contudo, podem ocorrer outros tipos de vazamentos de recurso. Por exemplo, arquivos, conexões de banco de dados e conexões de rede que não são fechadas corretamente depois de não serem mais necessárias podem não estar disponíveis para uso em outros programas.



#### Dica de prevenção de erro H.2

*Um problema sutil é que a linguagem Java não elimina totalmente os vazamentos de memória. Ela não fará a coleta de lixo de um objeto até que não restem mais referências a ele. Assim, se você erroneamente mantiver referências para objetos indesejados, poderão ocorrer vazamentos de memória. Para ajudar a evitar esse problema, configure as variáveis de tipo de referência como null quando não forem mais necessárias.*

O bloco finally (que consiste na palavra-chave finally, seguida por código incluído entre chaves), às vezes referido como **cláusula finally**, é opcional. Se estiver presente, ele é colocado após o último bloco catch. Se não houver blocos catch, o bloco finally virá imediatamente depois do bloco try.

O bloco finally será executado, seja lançada uma exceção ou não no bloco try correspondente. O bloco finally também será executado se um bloco try terminar com uma instrução return, break ou continue, ou simplesmente chegando à sua chave de fechamento. O bloco finally *não* será executado se o aplicativo for encerrado prematuramente a partir de um bloco try que chame o método **System.exit**. Esse método termina um aplicativo imediatamente.

Como um bloco finally quase sempre é executado, normalmente ele contém código de liberação de recursos. Suponha que um recurso seja alocado em um bloco try. Se não ocorre uma exceção, os blocos catch são pulados e o controle passa para o bloco finally, o qual libera o recurso. Então, o controle passa para a primeira instrução após o

bloco `finally`. Se ocorre uma exceção no bloco `try`, ele termina. Se o programa captura a exceção em um dos blocos `catch` correspondentes, ele processa a exceção e, então, o bloco `finally` libera o recurso e o controle passa para a primeira instrução após o bloco `finally`. Se o programa não captura a exceção, o bloco `finally` *ainda* libera o recurso e é feita uma tentativa de capturar a exceção em um método chamador.



### Dica de prevenção de erro H.3

*O bloco `finally` é o lugar ideal para liberar recursos adquiridos em um bloco `try` (como arquivos abertos), o que ajuda a eliminar vazamentos de recurso.*



### Dica de desempenho H.1

*Sempre libere um recurso explicitamente e o mais cedo possível, quando não for mais necessário. Isso torna os recursos disponíveis para reutilização assim que possível, aumentando assim a utilização de recursos.*

Se uma exceção ocorrida em um bloco `try` não pode ser capturada por uma das rotinas de tratamento `catch` do bloco `try`, o programa pula o restante do bloco `try` e o controle passa para o bloco `finally`. Então, o programa passa a exceção para o próximo bloco `try` externo – normalmente no método chamador –, onde um bloco `catch` associado poderá capturá-la. Esse processo pode ocorrer por muitos níveis de blocos `try`. Além disso, a exceção pode não ser capturada.

Se um bloco `catch` lança uma exceção, o bloco `finally` ainda é executado. Então, a exceção é passada para o próximo bloco `try` externo – outra vez, normalmente no método chamador.

A Figura H.3 demonstra que o bloco `finally` é executado mesmo que uma exceção não seja lançada no bloco `try` correspondente. O programa contém os métodos estáticos `main` (linhas 6 a 18), `throwException` (linhas 21 a 44) e `doesNotThrowException` (linhas 47 a 64). Os métodos `throwException` e `doesNotThrowException` são declarados como `static`, de modo que `main` pode chamá-los diretamente sem instanciar um objeto `UsingExceptions`.

```

1 // Fig. H.3: UsingExceptions.java
2 // Mecanismo de tratamento de exceção try...catch...finally.
3
4 public class UsingExceptions
5 {
6 public static void main(String[] args)
7 {
8 try
9 {
10 throwException(); // chama método throwException
11 } // fim do try
12 catch (Exception exception) // exceção lançada por throwException
13 {
14 System.err.println("Exception handled in main");
15 } // fim do catch
16
17 doesNotThrowException();
18 } // fim de main
19
20 // demonstra try...catch...finally

```

**Figura H.3** Mecanismo de tratamento de exceção `try...catch...finally`. (continua)

```

21 public static void throwException() throws Exception
22 {
23 try // lança uma exceção e a captura imediatamente
24 {
25 System.out.println("Method throwException");
26 throw new Exception(); // gera exceção
27 } // fim do try
28 catch (Exception exception) // captura exceção lançada no try
29 {
30 System.err.println(
31 "Exception handled in method throwException");
32 throw exception; // lança novamente para mais processamento
33
34 // este código não seria atingido; causaria erros de compilação
35 } // fim do catch
36 finally // executado independentemente do que ocorre em try...catch
37 {
38 System.err.println("Finally executed in throwException");
39 } // fim do finally
40
41 // este código não seria atingido; causaria erros de compilação
42 } // fim do método throwException
43
44 // demonstra finally quando não ocorre exceção
45 public static void doesNotThrowException()
46 {
47 try // o bloco try não lança uma exceção
48 {
49 System.out.println("Method doesNotThrowException");
50 } // fim do try
51 catch (Exception exception) // não é executado
52 {
53 System.err.println(exception);
54 } // fim do catch
55 finally // executado independentemente do que ocorre em try...catch
56 {
57 System.err.println(
58 "Finally executed in doesNotThrowException");
59 } // fim do finally
60
61 System.out.println("End of method doesNotThrowException");
62 } // fim do método doesNotThrowException
63 } // fim da classe UsingExceptions

```

```

Method throwException
Exception handled in method throwException
Finally executed in throwException
Exception handled in main
Method doesNotThrowException
Finally executed in doesNotThrowException
End of method doesNotThrowException

```

**Figura H.3** Mecanismo de tratamento de exceção try...catch...finally.

System.out e System.err são **fluxos** – sequências de bytes. Enquanto System.out (conhecido como **fluxo de saída padrão**) exibe a saída de um programa, System.err (conhecido como **fluxo de erro padrão**) exibe os erros de um programa. A saída desses fluxos pode ser redirecionada (isto é, enviada para algum outro lugar que não seja o

prompt de comando, como para um arquivo). O uso de dois fluxos diferentes permite separar facilmente as mensagens de erro de outra saída. Por exemplo, a saída de dados de `System.err` poderia ser enviada para um arquivo de log, enquanto a saída de dados de `System.out` poderia ser exibida na tela. Por simplicidade, este apêndice não vai redirecionar a saída de `System.err`, mas vai exibir tais mensagens no prompt de comando. Você vai aprender mais sobre fluxos no Apêndice J.

### Lançando exceções com a instrução `throw`

O método `main` (Fig. H.3) começa a ser executado, entra em seu bloco `try` e chama o método `throwException` (linha 10) imediatamente. O método `throwException` lança um objeto `Exception`. A instrução da linha 26 é conhecida como **instrução `throw`** – ela é executada para indicar que ocorreu uma exceção. Até aqui, você capturou apenas exceções lançadas por métodos chamados. Você mesmo pode lançar exceções, usando a instrução `throw`. Assim como as exceções lançadas pelos métodos da API Java, isso indica aos aplicativos clientes que ocorreu um erro. Uma instrução `throw` especifica um objeto a ser lançado. O operando de uma instrução `throw` pode ser de qualquer classe derivada da classe `Throwable`.



#### Observação sobre engenharia de software H.5

*Quando `toString` é chamado em qualquer objeto `Throwable`, a string resultante inclui a string descritiva que foi fornecida ao construtor ou simplesmente o nome de classe, caso não seja fornecida uma string.*



#### Observação sobre engenharia de software H.6

*Um objeto pode ser lançado sem conter informações sobre o problema ocorrido. Nesse caso, simplesmente saber que ocorreu um tipo de exceção em particular pode fornecer informações suficientes para a rotina de tratamento processar o problema corretamente.*



#### Observação sobre engenharia de software H.7

*As exceções podem ser lançadas a partir de construtores. Quando um erro é detectado em um construtor, uma exceção deve ser lançada para evitar a criação de um objeto incorretamente formado.*

### Relançando exceções

A linha 32 da Fig. H.3 **relança a exceção**. As exceções são relançadas quando um bloco `catch`, ao receber uma exceção, decide que não pode processá-la ou que só pode processá-la parcialmente. Relançar uma exceção transfere o tratamento de exceção (ou talvez de uma parte dela) a outro bloco `catch` associado a uma instrução `try` externa. Uma exceção é relançada com a **palavra-chave `throw`**, seguida de uma referência ao objeto exceção que acabou de ser capturado. As exceções não podem ser relançadas a partir de um bloco `finally`, pois o parâmetro de exceção (uma variável local) do bloco `catch` não existe mais.

Quando ocorre um relançamento, o *próximo bloco `try` circundante* detecta a exceção relançada e os blocos `catch` desse bloco `try` tentam tratá-la. Neste caso, o próximo bloco `try` circundante se encontra nas linhas 8 a 11, no método `main`. Contudo, antes que a exceção relançada seja tratada, o bloco `finally` (linhas 37 a 40) é



executado. Então, o método `main` detecta a exceção relançada no bloco `try` e a trata no bloco `catch` (linhas 12 a 15).

Em seguida, `main` chama o método `doesNotThrowException` (linha 17). Nenhuma exceção é lançada no bloco `try` de `doesNotThrowException` (linhas 49 a 52); portanto, o programa pula o bloco `catch` (linhas 53 a 56), mas o bloco `finally` (linha 57 a 61) é executado. O controle passa para a instrução após o bloco `finally` (linha 63). Então, o controle volta para `main` e o programa termina.



#### **Erro de programação comum H.6**

*Se uma exceção não tiver sido capturada quando o controle entrar em um bloco `finally` e esse bloco lançar uma exceção que não é capturada no bloco `finally`, a primeira exceção será perdida e a exceção do bloco `finally` será retornada para o método chamado.*



#### **Dica de prevenção de erro H.4**

*Evite colocar código que pode lançar uma exceção em um bloco `finally`. Se esse código for exigido, inclua-o em um `try...catch` dentro do bloco `finally`.*



#### **Erro de programação comum H.7**

*Supor que uma exceção lançada a partir de um bloco `catch` vai ser processada por esse bloco `catch` ou por qualquer outro bloco `catch` associado à mesma instrução `try` pode levar a erros de lógica.*



#### **Boa prática de programação H.1**

*O tratamento de exceção se destina a remover código de processamento de erro da linha principal do código de um programa a fim de melhorar a clareza do programa. Não coloque `try...catch...finally` em torno de cada instrução que pode lançar uma exceção. Isso dificulta a leitura dos programas. Em vez disso, coloque um bloco `try` em torno de uma parte significativa de seu código. Após esse bloco `try`, coloque blocos `catch` que tratem cada exceção possível e, após os blocos `catch`, coloque um único bloco `finally` (caso seja exigido).*

## **H.7 Stack unwinding e obtenção de informações de um objeto exceção**

Quando uma exceção é lançada, mas não capturada em um escopo específico, a pilha de chamada de métodos é “desenrolada” e é feita uma tentativa de capturar a exceção no próximo bloco `try` externo. Esse processo é chamado de **stack unwinding** (literalmente, desenrolar a pilha). Desenrolar a pilha de chamada de métodos significa que o método no qual a exceção não foi capturada *termina*, todas as variáveis locais desse método saem de escopo e o controle retorna para a instrução que chamou esse método originalmente. Se um bloco `try` engloba essa instrução, é feita uma tentativa de capturar a exceção. Se um bloco `try` não engloba essa instrução ou se a exceção não é capturada, o stack unwinding ocorre novamente. A Figura H.4 demonstra o stack unwinding, e a rotina de tratamento de exceção em `main` mostra como acessar os dados em um objeto de exceção.



```

1 // Fig. H.4: UsingExceptions.java
2 // Stack unwinding e obtenção de dados de um objeto exceção.
3
4 public class UsingExceptions
5 {
6 public static void main(String[] args)
7 {
8 try
9 {
10 method1(); // chama method1
11 } // fim do try
12 catch (Exception exception) // captura a exceção lançada em method1
13 {
14 System.err.printf("%s\n\n", exception.getMessage());
15 exception.printStackTrace(); // imprime o rastreamento de pilha de exceção
16
17 // obtém as informações do rastreamento de pilha
18 StackTraceElement[] traceElements = exception.getStackTrace();
19
20 System.out.println("\nStack trace from getStackTrace:");
21 System.out.println("Class\t\tFile\t\t\tLine\t\tMethod");
22
23 // faz loop pelos traceElements para obter a descrição da exceção
24 for (StackTraceElement element : traceElements)
25 {
26 System.out.printf("%s\t", element.getClassName());
27 System.out.printf("%s\t", element.getFileName());
28 System.out.printf("%s\t", element.getLineNumber());
29 System.out.printf("%s\n", element.getMethodName());
30 } // fim do for
31 } // fim do catch
32 } // fim de main
33
34 // chama method2; lança exceções de volta para main
35 public static void method1() throws Exception
36 {
37 method2();
38 } // fim do método method1
39
40 // chama method3; lança exceções de volta para method1
41 public static void method2() throws Exception
42 {
43 method3();
44 } // fim do método method2
45
46 // lança objeto Exception de volta para method2
47 public static void method3() throws Exception
48 {
49 throw new Exception("Exception thrown in method3");
50 } // fim do método method3
51 } // fim da classe UsingExceptions

```

Exception thrown in method3

```

java.lang.Exception: Exception thrown in method3
 at UsingExceptions.method3(UsingExceptions.java:49)
 at UsingExceptions.method2(UsingExceptions.java:43)
 at UsingExceptions.method1(UsingExceptions.java:37)
 at UsingExceptions.main(UsingExceptions.java:10)

```

Stack trace from getStackTrace:

| Class           | File                 | Line | Method  |
|-----------------|----------------------|------|---------|
| UsingExceptions | UsingExceptions.java | 49   | method3 |
| UsingExceptions | UsingExceptions.java | 43   | method2 |
| UsingExceptions | UsingExceptions.java | 37   | method1 |
| UsingExceptions | UsingExceptions.java | 10   | main    |

**Figura H.4** Stack unwinding e obtenção de dados de um objeto de exceção.

### Stack Unwinding

Em `main`, o bloco `try` (linhas 8 a 11) chama `method1` (declarado nas linhas 35 a 38), o qual, por sua vez, chama `method2` (declarado nas linhas 41 a 44), o qual, por sua vez, chama `method3` (declarado nas linhas 47 a 50). A linha 49 de `method3` lança um objeto `Exception` – esse é o *ponto de lançamento*. Como a instrução `throw` na linha 49 *não* está englobada em um bloco `try`, o *stack unwinding* ocorre – `method3` termina na linha 49 e, então, retorna o controle para a instrução de `method2` que chamou `method3` (ou seja, a linha 43). Como *nenhum* bloco `try` engloba a linha 43, o *stack unwinding* ocorre novamente – `method2` termina na linha 43 e retorna o controle para a instrução de `method1` que chamou `method2` (ou seja, a linha 37). Como *nenhum* bloco `try` engloba a linha 37, o *stack unwinding* ocorre mais uma vez – `method1` termina na linha 37 e retorna o controle para a instrução de `main` que chamou `method1` (ou seja, a linha 10). O bloco `try` nas linhas 8 a 11 engloba essa instrução. A exceção não foi tratada, de modo que o bloco `try` termina e o primeiro bloco `catch` correspondente (linhas 12 a 31) captura e processa a exceção. Se nenhum bloco `catch` correspondesse e a exceção não fosse declarada em cada método que a lança, ocorreria um erro de compilação. Lembre-se de que nem sempre isso acontece – para exceções *não verificadas*, o aplicativo compilará, mas será executado com resultados inesperados.

### Obtendo dados de um objeto de exceção

Lembre-se de que as exceções derivam da classe `Throwable`. A classe `Throwable` oferece o método `printStackTrace`, o qual gera o rastreamento de pilha (discutido na seção H.2) no fluxo de erro padrão. Frequentemente, isso é útil em teste e depuração. A classe `Throwable` fornece também o método `getStackTrace`, o qual recupera informações de rastreamento de pilha que podem ser impressas por `printStackTrace`. O método `getMessage` da classe `Throwable` retorna a string descritiva armazenada em uma exceção.



#### Dica de prevenção de erro H.5

*Uma exceção que não é capturada em um aplicativo faz a rotina de tratamento de exceção padrão da linguagem Java ser executada. Ela exibe o nome da exceção, uma mensagem descritiva indicando o problema ocorrido e um rastreamento de pilha de execução completo. Em um aplicativo com apenas uma thread de execução, o aplicativo termina. Em um aplicativo com várias threads, a thread que causou a exceção termina.*



#### Dica de prevenção de erro H.6

*O método `toString` de `Throwable` (herdado por todas as subclasses de `Throwable`) retorna uma String contendo o nome da classe da exceção e uma mensagem descritiva.*

A rotina de tratamento `catch` da Fig. H.4 (linhas 12 a 31) demonstra `getMessage`, `printStackTrace` e `getStackTrace`. Se quiséssemos gerar as informações de rastreamento de pilha em fluxos que não o fluxo de erro padrão, poderíamos usar as informações retornadas de `getStackTrace` e gerá-las na saída de outro fluxo ou usar uma das versões sobrecarregadas do método `printStackTrace`.

A linha 14 chama o método `getMessage` da exceção para obter a descrição da exceção. A linha 15 chama o método `printStackTrace` da exceção para gerar o rastreamento de pilha que indica onde a exceção ocorreu. A linha 18 chama o método `getStackTrace` da exceção para obter as informações de rastreamento de pilha como um array de objetos `StackTraceElement`. As linhas 24 a 30 obtêm cada `StackTraceElement` do array e chamam seus métodos `getClassName`, `getFileName`, `getLineNumber` e `getMethodName` para obter, respectivamente, o nome da classe, o nome do arquivo, o número da linha e o nome do

método desse `StackTraceElement`. Cada `StackTraceElement` representa uma chamada de método na pilha de chamadas de método.

A saída do programa mostra que as informações de rastreamento de pilha impressas por `printStackTrace` seguem o padrão *nomeDaClasse.nomeDoMétodo(nomeDoArquivo:númeroDaLinha)*, onde *nomeDaClasse*, *nomeDoMétodo* e *nomeDoArquivo* indicam, respectivamente, os nomes da classe, do método e do arquivo em que a exceção ocorreu, e *númeroDaLinha* indica onde no arquivo a exceção ocorreu. Você viu isso na saída da Fig. H.1. O método `getStackTrace` permite um processamento personalizado das informações da exceção. Compare a saída de `printStackTrace` com a saída criada a partir dos `StackTraceElements` para ver que ambas contêm as mesmas informações de rastreamento de pilha.



### Observação sobre engenharia de software H.8

*Nunca forneça uma rotina de tratamento catch com um corpo vazio – na verdade, isso vai ignorar a exceção. No mínimo, use `printStackTrace` para gerar uma mensagem de erro a fim de indicar que existe um problema.*

## H.8 Para finalizar

Neste apêndice, você aprendeu a usar tratamento de exceção para lidar com erros. Aprendeu que ele permite remover código de tratamento de erro da “linha principal” da execução do programa. Mostramos como usar blocos `try` para englobar código que pode lançar uma exceção e como usar blocos `catch` para lidar com as exceções que podem surgir. Você conheceu o modelo de terminação de tratamento de exceção, o qual prescreve que, depois de uma exceção ser tratada, o controle do programa não volta para o ponto de lançamento. Discutimos as exceções verificadas *versus* não verificadas e como especificar, com a cláusula `throws`, as exceções que um método pode lançar. Você aprendeu a usar o bloco `finally` para liberar recursos, ocorra uma exceção ou não. Aprendeu também a lançar e relançar exceções. Mostramos como obter informações sobre uma exceção usando os métodos `printStackTrace`, `getStackTrace` e `getMessage`. No próximo apêndice, discutimos os conceitos de interface gráfica do usuário e explicamos os fundamentos do tratamento de eventos.

## Exercícios de revisão

- H.1** Liste cinco exemplos comuns de exceção.
- H.2** Cite vários motivos pelos quais as técnicas de tratamento de exceção não devem ser usadas para controle de programa convencional.
- H.3** Por que as exceções são particularmente adequadas para lidar com erros produzidos por métodos das classes da API Java?
- H.4** O que é “vazamento de recurso”?
- H.5** Se nenhuma exceção é lançada em um bloco `try`, para onde vai o controle quando termina a execução do bloco `try`?
- H.6** Cite a principal vantagem de usar `catch( Exception nomeDaExceção )`.
- H.7** Um aplicativo convencional deve capturar objetos `Error`? Explique.

- H.8** O que acontece se nenhuma rotina de tratamento `catch` corresponde ao tipo de um objeto lançado?
- H.9** O que acontece se vários blocos `catch` correspondem ao tipo do objeto lançado?
- H.10** Por que um programador especificaria um tipo de superclasse como tipo em um bloco `catch`?
- H.11** Qual é o principal motivo para usar blocos `finally`?
- H.12** O que acontece quando um bloco `catch` lança um objeto `Exception`?
- H.13** O que a instrução `throw` *referênciaDeExceção* faz em um bloco `catch`?
- H.14** O que acontece com uma referência local em um bloco `try` quando esse bloco lança um objeto `Exception`?

## Respostas dos exercícios de revisão

- H.1** Esgotamento da memória, índice de array fora dos limites, estouro aritmético, divisão por zero, parâmetros de método inválidos.
- H.2** (a) O tratamento de exceção é feito para lidar com situações que raramente ocorrem, mas que muitas vezes resultam no término do programa, e não para lidar com situações que surgem o tempo todo. (b) Com estruturas de controle convencionais, o fluxo de controle geralmente é mais claro e eficiente do que com exceções. (c) As exceções adicionais podem interferir nas exceções do tipo erro genuínas. Torna-se mais difícil monitorar o maior número de casos de exceção.
- H.3** É improvável que métodos de classes da API Java realizem processamento de erro que satisfaça as necessidades exclusivas de todos os usuários.
- H.4** Um “vazamento de recurso” ocorre quando um programa que está em execução não libera um recurso corretamente quando ele não é mais necessário.
- H.5** Os blocos `catch` dessa instrução `try` são pulados e o programa retoma a execução após o último bloco `catch`. Se existe um bloco `finally`, ele é executado primeiro; então, o programa retoma a execução após o bloco `finally`.
- H.6** A forma `catch( Exception nomeDaExceção)` captura qualquer tipo de exceção lançada em um bloco `try`. Uma vantagem é que nenhum objeto `Exception` lançado pode passar sem ser capturado. Você pode então decidir se vai tratar a exceção ou se possivelmente vai relançá-la.
- H.7** `Errors` normalmente são problemas sérios no sistema Java subjacente; a maioria dos programas não vai querer capturar objetos `Error`, pois não poderá se recuperar deles.
- H.8** Isso faz que a busca por uma correspondência continue na próxima instrução `try` circundante. Se existir um bloco `finally`, ele será executado antes que a exceção passe para a próxima instrução `try` circundante. Se não houver uma instrução `try` circundante para a qual existam blocos `catch` correspondentes e as exceções sejam declaradas (ou não verificadas), será impresso um rastreamento de pilha e a thread atual terminará antecipadamente. Se as exceções são verificadas, mas não capturadas ou declaradas, ocorrem erros de compilação.
- H.9** O primeiro bloco `catch` correspondente após o bloco `try` é executado.
- H.10** Isso permitiria a um programa capturar tipos relacionados de exceções e processá-las de maneira uniforme. Contudo, muitas vezes é útil processar os tipos de subclasse individualmente para ter um tratamento de exceção mais preciso.
- H.11** O bloco `finally` é a maneira preferida para liberar recursos a fim de evitar vazamentos de recurso.

- H.12** Primeiramente, o controle passa para o bloco `finally`, caso haja um. Então, a exceção será processada por um bloco `catch` (se existir um) associado a um bloco `try` circundante (se existir um).
- H.13** Ela relança a exceção para processamento por parte de uma rotina de tratamento de exceção de uma instrução `try` circundante, após a execução do bloco `finally` da instrução `try` atual.
- H.14** A referência sai do escopo. Se o objeto referenciado se torna inatingível, pode ser marcado para coleta de lixo.

## Exercícios

- H.15** (*Condições excepcionais*) Liste as várias condições excepcionais que ocorreram nos programas dos apêndices até o momento. Liste o máximo de condições excepcionais adicionais que você puder. Para cada uma delas, descreva brevemente como um programa trataria a exceção utilizando as técnicas de tratamento discutidas neste apêndice. Exceções típicas incluem a divisão por zero e o índice de array fora dos limites.
- H.16** (*Exceções e falha do construtor*) Até este apêndice, achamos um pouco complicado lidar com erros detectados por construtores. Explique por que o tratamento de exceções é uma maneira eficiente de lidar com falhas de construtor.
- H.17** (*Capturando exceções com superclasses*) Use herança para criar uma superclasse de exceção (chamada `ExceptionA`) e as subclasses de exceção `ExceptionB` e `ExceptionC`, onde `ExceptionB` herda de `ExceptionA` e `ExceptionC` herda de `ExceptionB`. Escreva um programa para demonstrar que o bloco `catch` do tipo `ExceptionA` captura exceções dos tipos `ExceptionB` e `ExceptionC`.
- H.18** (*Capturando exceções com a classe `Exception`*) Escreva um programa que demonstre como várias exceções são capturadas com
 

```
catch (Exception exception)
```

Desta vez, defina as classes `ExceptionA` (que herda da classe `Exception`) e `ExceptionB` (que herda da classe `ExceptionA`). Em seu programa, crie blocos `try` que lancem exceções dos tipos `ExceptionA`, `ExceptionB`, `NullPointerException` e `IOException`. Todas as exceções devem ser capturadas com blocos `catch` especificando o tipo `Exception`.
- H.19** (*Ordem dos blocos `catch`*) Escreva um programa que mostre que a ordem dos blocos `catch` é importante. Se você tentar capturar um tipo de exceção de superclasse antes de um tipo de subclasse, o compilador deve gerar erros.
- H.20** (*Falha de construtor*) Escreva um programa que mostre um construtor passando informações sobre falha de construtor para uma rotina de tratamento de exceção. Defina a classe `SomeClass`, a qual lança um objeto `Exception` no construtor. Seu programa deve tentar criar um objeto de tipo `SomeClass` e capturar a exceção lançada a partir do construtor.
- H.21** (*Relançando exceções*) Escreva um programa que ilustre o relançamento de uma exceção. Defina os métodos `someMethod` e `someMethod2`. O método `someMethod2` deve inicialmente lançar uma exceção. O método `someMethod` deve chamar `someMethod2`, capturar a exceção e relançá-la. Chame `someMethod` a partir do método `main` e capture a exceção relançada. Imprima o rastreamento de pilha dessa exceção.
- H.22** (*Capturando exceções com escopos externos*) Escreva um programa que mostre que um método com seu próprio bloco `try` não precisa capturar todo erro possível gerado dentro do `try`. Algumas exceções podem se infiltrar e ser tratadas em outros escopos.

# Componentes de interface gráfica do usuário e tratamento de eventos

# I

## Objetivos

Neste capítulo, você vai:

- Aprender a usar a aparência e o comportamento (look-and-feel) independente de plataforma Nimbus do Java.
- Construir interfaces gráficas e tratar eventos gerados por interações do usuário com as interfaces gráficas.
- Usar classes aninhadas e classes internas anônimas para implementar rotinas de tratamento de evento.



- |                                                                                            |                                                                                    |
|--------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| <b>I.1</b> Introdução                                                                      | <b>I.5</b> Como funciona o tratamento de eventos                                   |
| <b>I.2</b> Aparência e comportamento Nimbus                                                | <b>I.6</b> JButton                                                                 |
| <b>I.3</b> Campos de texto e uma introdução ao tratamento de eventos com classes aninhadas | <b>I.7</b> JComboBox: uso de uma classe interna anônima para tratamento de eventos |
| <b>I.4</b> Tipos de evento comuns em interfaces gráficas e interfaces receptoras           | <b>I.8</b> Classes adaptadoras                                                     |
|                                                                                            | <b>I.9</b> Para finalizar                                                          |

*Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios*

## I.1 Introdução

Uma **interface gráfica do usuário** (GUI) apresenta um mecanismo amigável para interagir com um aplicativo. Ela proporciona “aparência e comportamento” (look-and feel) singulares ao aplicativo. As interfaces de usuário são construídas a partir de **componentes**, como rótulos, botões, caixas de texto, menus, barras de rolagem e muitos outros. Às vezes, eles são chamados de controles ou widgets – abreviação de *window gadgets* (artefatos de janela). Um componente de interface gráfica do usuário é um objeto com o qual o usuário interage por meio do mouse, do teclado ou outra forma de entrada, como o reconhecimento de voz. Neste apêndice, apresentamos alguns componentes de interface gráfica do usuário básicos e como responder às interações do usuário com eles – uma técnica conhecida como tratamento de evento. Discutimos também as *classes aninhadas* e as *classes internas anônimas*, que são comumente utilizadas para tratamento de evento em aplicativos Java e Android.

## I.2 Aparência e comportamento Nimbus

Em nossas capturas de tela, usamos a elegante aparência e comportamento (look-and-feel) independente de plataforma **Nimbus** do Java. Você pode usar Nimbus de três maneiras:

1. Configurá-lo como padrão para todos os aplicativos Java executados em seu computador.
2. Configurá-lo como aparência e comportamento no momento de ativar um aplicativo passando um argumento de linha de comando para o comando java.
3. Configurá-lo como aparência e comportamento por meio de programação em seu aplicativo.

Configuramos Nimbus como padrão para todos os aplicativos Java. Para isso, você precisa criar um arquivo de texto chamado `swing.properties` na pasta `lib` da pasta de instalação de seu JDK e na pasta de instalação de seu JRE. Coloque a seguinte linha de código no arquivo:

```
swing.defaultlaf=com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel
```

Para obter mais informações sobre como localizar essas pastas de instalação, visite

```
bit.ly/JavaInstallationInstructions
```

Além do JRE independente, há um JRE aninhado na pasta de instalação de seu JDK. Se estiver usando um IDE que depende do JDK, talvez também seja necessário colocar o arquivo `swing.properties` na pasta `lib` da pasta `jre` aninhada.

Se preferir selecionar Nimbus aplicativo por aplicativo, coloque o seguinte argumento de linha de comando após o comando java e antes do nome do aplicativo ao executar o aplicativo:

```
-Dswing.defaultlaf=com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel
```

### I.3 Campos de texto e uma introdução ao tratamento de eventos com classes aninhadas

Normalmente, um usuário interage com a interface gráfica de um aplicativo para indicar as tarefas que este deve executar. Por exemplo, quando você escreve um e-mail em um aplicativo de correio eletrônico, clicar no botão **Enviar** diz ao aplicativo para que envie o e-mail para o endereço especificado. As interfaces gráficas do usuário são **baseadas em eventos**. Quando o usuário interage com um componente de interface gráfica, a interação – conhecida como **evento** – impele o programa a executar a tarefa. Algumas interações de usuário comuns, que fazem um aplicativo executar uma tarefa, incluem clicar em um botão, digitar em um campo de texto, selecionar um item em um menu, fechar uma janela e mover o mouse. O código que executa uma tarefa em resposta a um evento é chamado de **rotina de tratamento de evento** e o processo global de responder aos eventos é conhecido como **tratamento de evento**.

Vamos considerar dois componentes de interface gráfica do usuário que podem gerar eventos – `JTextField` e `JPasswordField` (pacote `javax.swing`). A classe `JTextField` estende a classe `JTextComponent` (pacote `javax.swing.text`), a qual fornece muitos recursos comuns para componentes baseados em texto do Swing. A classe `JPasswordField` estende `JTextField` e acrescenta métodos específicos para processamento de senhas. Cada um desses componentes é uma área de apenas uma linha na qual o usuário pode digitar texto por meio do teclado. Os aplicativos também podem exibir texto em um componente `JTextField` (veja a saída da Fig. I.2). Um componente `JPasswordField` mostra os caracteres que estão sendo digitados à medida que o usuário os insere, mas oculta os caracteres reais com um **caractere de eco**, supondo que representem uma senha que deva ser conhecida apenas pelo usuário.

Quando o usuário digita em um componente `JTextField` ou `JPasswordField` e pressiona *Enter*, ocorre um evento. Nosso próximo exemplo demonstra como um programa pode executar uma tarefa em resposta a esse evento. As técnicas mostradas aqui são aplicáveis a todos os componentes de interface gráfica do usuário que geram eventos.

O aplicativo das Figs. I.1 e I.2 usa as classes `JTextField` e `JPasswordField` para criar e manipular quatro campos de texto. Quando o usuário digita em um dos campos de texto e pressiona *Enter*, o aplicativo exibe uma caixa de diálogo de mensagem contendo o texto digitado. Você pode digitar apenas no campo de texto que está “no **foco**”. Quando você clica em um componente, ele *recebe o foco*. Isso é importante, pois o campo de texto que tem o foco é o que gera um evento ao se pressionar *Enter*. Neste exemplo, quando você pressiona *Enter* no componente `JPasswordField`, a senha é revelada. Começamos discutindo a configuração da interface gráfica do usuário e, então, discutimos o código de tratamento de evento.



```

1 // Fig. I.1: TextFieldFrame.java
2 // Componentes JTextField e JPasswordField.
3 import java.awt.FlowLayout;
4 import java.awt.event.ActionListener;
5 import java.awt.event.ActionEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JTextField;
8 import javax.swing.JPasswordField;
9 import javax.swing.JOptionPane;
10
11 public class TextFieldFrame extends JFrame
12 {
13 private JTextField textField1; // campo de texto com tamanho configurado
14 private JTextField textField2; // campo de texto construído com texto
15 private JTextField textField3; // campo de texto com texto e tamanho
16 private JPasswordField passwordField; // campo de senha com texto
17
18 // o construtor de TextFieldFrame adiciona componentes JTextField a JFrame
19 public TextFieldFrame()
20 {
21 super("Testing JTextField and JPasswordField");
22 setLayout(new FlowLayout()); // configura o layout do quadro
23
24 // constrói campo de texto com 10 colunas
25 textField1 = new JTextField(10);
26 add(textField1); // adiciona textField1 a JFrame
27
28 // constrói campo de texto com texto padrão
29 textField2 = new JTextField("Enter text here");
30 add(textField2); // adiciona textField2 a JFrame
31
32 // constrói campo de texto com texto padrão e 21 colunas
33 textField3 = new JTextField("Uneditable text field", 21);
34 textField3.setEditable(false); // desabilita a edição
35 add(textField3); // adiciona textField3 a JFrame
36
37 // constrói campo de senha com texto padrão
38 passwordField = new JPasswordField("Hidden text");
39 add(passwordField); // adiciona passwordField a JFrame
40
41 // registra rotinas de tratamento de evento
42 TextFieldHandler handler = new TextFieldHandler();
43 textField1.addActionListener(handler);
44 textField2.addActionListener(handler);
45 textField3.addActionListener(handler);
46 passwordField.addActionListener(handler);
47 } // fim do construtor de TextFieldFrame
48
49 // classe interna private para tratamento de eventos
50 private class TextFieldHandler implements ActionListener
51 {
52 // processa eventos de campo de texto
53 public void actionPerformed(ActionEvent event)
54 {
55 String string = ""; // declara string a exibir
56
57 // o usuário pressionou Enter no componente JTextField textField1
58 if (event.getSource() == textField1)
59 string = String.format("textField1: %s",

```

**Figura I.1** Componentes JTextField e JPasswordField.

```

60 event.getActionCommand());
61
62 // o usuário pressionou Enter no componente JTextField textField2
63 else if (event.getSource() == textField2)
64 string = String.format("textField2: %s",
65 event.getActionCommand());
66
67 // o usuário pressionou Enter no componente JTextField textField3
68 else if (event.getSource() == textField3)
69 string = String.format("textField3: %s",
70 event.getActionCommand());
71
72 // o usuário pressionou Enter no componente JTextField passwordField
73 else if (event.getSource() == passwordField)
74 string = String.format("passwordField: %s",
75 event.getActionCommand());
76
77 // exibe o conteúdo de JTextField
78 JOptionPane.showMessageDialog(null, string);
79 } // fim do método actionPerformed
80 } // fim da classe interna private TextFieldHandler
81 } // fim da classe TextFieldFrame

```

**Figura I.1** Componentes JTextField e JPasswordField.

As linhas 3 a 9 importam as classes e interfaces utilizadas neste exemplo. A classe `TextFieldFrame` estende `JFrame` e declara três variáveis de `JTextField` e uma variável de `JPasswordField` (linhas 13 a 16). Cada um dos campos de texto correspondente é instanciado e anexado a `TextFieldFrame` no construtor (linhas 19 a 47).

### **Especificando o layout**

Ao construir uma interface gráfica do usuário, você deve anexar cada componente da interface a um contêiner, como uma janela criada com `JFrame`. Além disso, normalmente você precisa decidir *onde* vai posicionar cada componente da interface gráfica – o que é conhecido como especificar o layout. A linguagem Java fornece vários **gerenciadores de layout** que podem ajudá-lo a posicionar componentes.

Muitos IDEs fornecem ferramentas de projeto de interface gráfica do usuário, nas quais é possível especificar visualmente tamanhos e posições exatos dos componentes usando o mouse; então, o IDE gerará o código da interface gráfica do usuário para você. Esses IDEs podem simplificar bastante a criação de interfaces gráficas do usuário.

Para garantirmos que nossas interfaces gráficas possam ser usadas com *qualquer* IDE, *não* utilizamos um IDE para criar o código da interface. Usamos gerenciadores de layout do Java para dimensionar e posicionar componentes. Com o gerenciador de layout **FlowLayout**, os componentes são colocados da esquerda para a direita em um contêiner, na ordem em que são adicionados. Quando não cabem mais componentes na linha atual, eles continuam a aparecer na linha seguinte, da esquerda para a direita. Se o contêiner é redimensionado, um elemento `FlowLayout` *reflui* os componentes, possivelmente com mais ou menos linhas, de acordo com a largura do novo contêiner. Todo contêiner tem um layout padrão, o qual estamos mudando de `TextFieldFrame` para `FlowLayout` (linha 22). O método **setLayout** é herdado indiretamente da classe `Container` para a classe `TextFieldFrame`. O argumento do método deve ser um objeto de uma classe que implemente a interface `LayoutManager` (por exemplo, `FlowLayout`). A linha 22 cria um novo objeto `FlowLayout` e passa sua referência como argumento para `setLayout`.

### ***Criando a interface gráfica do usuário***

A linha 25 cria `textField1` com 10 colunas de texto. A largura da coluna de texto em *pixels* é determinada pela largura média de um caractere na fonte atual do campo de texto. Quando o texto exibido em um campo de texto é mais largo do que o campo, uma parte do texto não é visível no lado direito. Se você tenta digitar em um campo de texto e o cursor chega à margem direita, o texto da margem esquerda é empurrado para o lado esquerdo do campo e não fica mais visível. Os usuários podem utilizar as teclas de seta à esquerda e à direita para moverem-se pelo texto inteiro. A linha 26 adiciona `textField1` a `JFrame`.

A linha 29 cria `textField2` com o texto inicial "Enter text here" a ser exibido no campo de texto. A largura do campo é determinada pela largura do texto padrão especificada no construtor. A linha 30 adiciona `textField2` a `JFrame`.

A linha 33 cria `textField3` e chama o construtor de `JTextField` com dois argumentos – o texto padrão "Uneditable text field" a ser exibido e a largura do campo de texto em colunas (21). A linha 34 usa o método `setEditable` (herdado da classe `JTextComponent` por `JTextField`) para tornar o campo de texto *não editável* – isto é, o usuário não pode modificar o texto no campo. A linha 35 adiciona `textField3` a `JFrame`.

A linha 38 cria `passwordField` com o texto "Hidden text" a ser exibido no campo de texto. A largura do campo é determinada pela largura do texto padrão. Quando executar o aplicativo, observe que o texto é exibido como uma série de asteriscos. A linha 39 adiciona `passwordField` a `JFrame`.

### ***Etapas exigidas para configurar o tratamento de evento para um componente de interface gráfica do usuário***

Este exemplo deve exibir uma caixa de diálogo de mensagem contendo o texto de um campo de texto quando o usuário pressionar *Enter* nesse campo. Antes que um aplicativo possa responder a um evento de um componente de interface gráfica do usuário em particular, você deve:

1. Criar uma classe que represente a rotina de tratamento de evento e que implemente uma interface adequada – conhecida como **interface receptora de eventos**.
2. Indicar que um objeto da classe da *etapa 1* deve ser notificado quando o evento ocorrer – conhecido como **registrar a rotina de tratamento de evento**.

### ***Usando uma classe aninhada para implementar uma rotina de tratamento de evento***

Todas as classes discutidas até agora eram as assim chamadas **classes de nível superior** – ou seja, não eram declaradas dentro de outra classe. A linguagem Java permite declarar classes *dentro* de outras classes – são as chamadas **classes aninhadas**. As classes aninhadas podem ser estáticas ou não estáticas. As classes aninhadas não estáticas são denominadas **classes internas** e são frequentemente usadas para implementar *rotinas de tratamento de evento*.

Um objeto da classe interna deve ser criado por um objeto da classe de nível superior que contenha a classe interna. Cada objeto de classe interna tem *implicitamente* uma referência para um objeto de sua classe de nível superior. O objeto de classe interna pode usar essa referência implícita para acessar diretamente todas as variáveis e métodos da classe de nível superior. Uma classe aninhada estática não exige um objeto de sua classe de nível superior e não tem implicitamente uma referência para um objeto da classe de nível superior.

### Classe aninhada *TextFieldHandler*

O tratamento de evento deste exemplo é feito por um objeto da classe interna *private TextFieldHandler* (linhas 50 a 80). Essa classe é *private* porque vai ser usada apenas para criar rotinas de tratamento de evento para os campos de texto da classe de nível superior *TextFieldFrame*. Como outros membros de classe, as *classes internas* podem ser declaradas como *public*, *protected* ou *private*. Como as rotinas de tratamento de evento tendem a ser específicas para o aplicativo no qual são definidas, frequentemente são implementadas como classes internas *private* ou como *classes internas anônimas* (seção I.7).

Os componentes de uma interface gráfica podem gerar muitos eventos em resposta às interações do usuário. Cada evento é representado por uma classe e pode ser processado somente pelo tipo de rotina de tratamento de evento apropriado. Normalmente, os eventos suportados por um componente são descritos na documentação da API Java da classe desse componente e suas superclasses. Quando o usuário pressiona *Enter* em um componente *TextField* ou *PasswordField*, ocorre um evento **ActionEvent** (pacote *java.awt.event*). Esse evento é processado por um objeto que implementa a interface **ActionListener** (pacote *java.awt.event*). As informações discutidas aqui estão disponíveis na documentação da API Java para as classes *TextField* e *ActionEvent*. Como *PasswordField* é uma subclasse de *TextField*, *PasswordField* suporta os mesmos eventos.

A fim de se preparar para tratar os eventos deste exemplo, a classe interna *TextFieldHandler* implementa a interface *ActionListener* e declara o único método dessa interface – *actionPerformed* (linhas 53 a 79). Esse método especifica as tarefas a serem executadas quando um evento *ActionEvent* ocorrer. Assim, a classe interna *TextFieldHandler* satisfaz a *etapa 1* listada anteriormente nesta seção. Vamos discutir os detalhes do método *actionPerformed* em breve.

### Registrando a rotina de tratamento de evento para cada campo de texto

No construtor de *TextFieldFrame*, a linha 42 cria um objeto *TextFieldHandler* e o atribui à variável *handler*. O método *actionPerformed* desse objeto vai ser chamado automaticamente quando o usuário pressionar *Enter* em qualquer um dos campos de texto da interface gráfica. Contudo, antes que isso possa ocorrer, o programa precisa registrar esse objeto como rotina de tratamento de evento para cada campo de texto. As linhas 43 a 46 são as instruções de registro de evento que especificam *handler* como rotina de tratamento de evento para os três componentes *TextField* e para *PasswordField*. O aplicativo chama o método **addActionListener** de *TextField* para registrar a rotina de tratamento de evento para cada componente. Esse método recebe como argumento um objeto *ActionListener*, o qual pode ser de qualquer classe que implemente *ActionListener*. O objeto *handler* é um *ActionListener*, pois a classe *TextFieldHandler* implementa *ActionListener*. Após a execução das linhas 43 a 46, o objeto *handler* **recebe eventos**. Agora, quando o usuário pressiona *Enter* em qualquer um desses quatro campos de texto, o método *actionPerformed* (linhas 53 a 79) da classe *TextFieldHandler* é chamado para tratar o evento. Se não for registrada uma rotina de tratamento de evento para um campo de texto em particular, o evento que ocorrer quando o usuário pressionar *Enter* nesse campo será **consumido** – isto é, será simplesmente ignorado pelo aplicativo.



#### Observação sobre engenharia de software I.1

O receptor de um evento deve implementar a interface receptora de eventos apropriada.



### Erro de programação comum I.1

*Esquecer-se de registrar um objeto rotina de tratamento de evento para um tipo de evento de um componente da interface gráfica do usuário em particular faz que eventos desse tipo sejam ignorados.*

### Detalhes do método `actionPerformed` da classe `TextFieldHandler`

Neste exemplo, estamos usando o método `actionPerformed` de um objeto de tratamento de evento (linhas 53 a 79) para tratar os eventos gerados pelos quatro campos de texto. Como queremos gerar o nome da variável de instância de cada campo de texto para propósitos de demonstração, precisamos determinar qual campo de texto gera o evento sempre que `actionPerformed` é chamado. A **origem do evento** é o componente da interface gráfica com a qual o usuário interagiu. Quando o usuário pressiona *Enter* enquanto um dos campos de texto ou o campo de senha *tem o foco*, o sistema cria um objeto `ActionEvent` único contendo informações sobre o evento que acabou de ocorrer, como a origem do evento e o texto que está no campo de texto. O sistema passa esse objeto `ActionEvent` para o método `actionPerformed` do receptor de eventos. A linha 55 declara a `String` que será exibida. A variável é inicializada com a **string vazia** – uma `String` que não contém caractere. O compilador exige que a variável seja inicializada, para o caso de nenhum dos desvios do `if` aninhado, nas linhas 58 a 75, ser executado.

O método `getSource` de `ActionEvent` (chamado nas linhas 58, 63, 68 e 73) retorna uma referência para a origem do evento. A condição na linha 58 pergunta: “A origem do evento é `textField1`?”. Essa condição compara referências com o operador `==` para determinar se se referem ao mesmo objeto. Se *ambas* se referem a `textField1`, o usuário pressionou *Enter* em `textField1`. Então, as linhas 59 e 60 criam uma `String` contendo a mensagem que a linha 78 exibe em uma caixa de diálogo. A linha 60 usa o método `getActionCommand` de `ActionEvent` para obter o texto digitado pelo usuário no campo de texto que gerou o evento.

Neste exemplo, exibimos o texto da senha no componente `JPasswordField` quando o usuário pressiona *Enter* nesse campo. Às vezes, é necessário processar os caracteres de uma senha por meio de programa. O método `getPassword` da classe `JPasswordField` retorna os caracteres da senha como um array de tipo `char`.

### Classe `TextFieldTest`

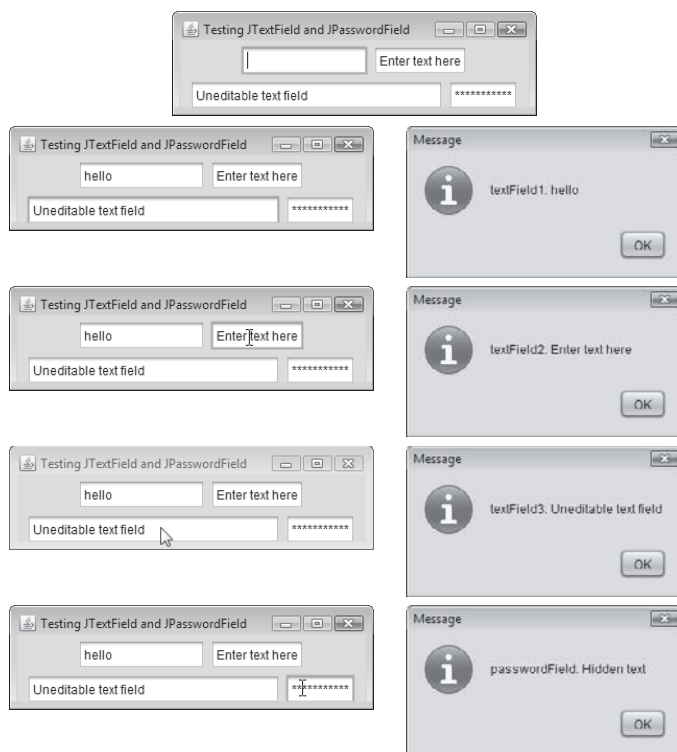
A classe `TextFieldTest` (Fig. I.2) contém o método `main` que executa este aplicativo e exibe um objeto da classe `TextFieldFrame`. Quando você executa o aplicativo, até o componente `JTextField` não editável (`textField3`) pode gerar um evento `ActionEvent`. Para testar isso, clique no campo de texto a fim de dar o foco a ele e, então, pressione *Enter*. Além disso, o texto da senha é exibido quando você pressiona *Enter* no componente `JPasswordField`. É claro que, normalmente, você não exibiria a senha!

Este aplicativo usou um único objeto da classe `TextFieldHandler` como receptor de evento para quatro campos de texto. É possível declarar vários objetos receptores de evento do mesmo tipo e registrar cada objeto para um evento separado do componente da interface gráfica do usuário. Essa técnica nos permite eliminar a lógica `if...else` utilizada na rotina de tratamento de eventos deste exemplo, fornecendo rotinas separadas para os eventos de cada componente.

```

1 // Fig. I.2: TextFieldTest.java
2 // Testando TextFieldFrame.
3 import javax.swing.JFrame;
4
5 public class TextFieldTest
6 {
7 public static void main(String[] args)
8 {
9 TextFieldFrame textFieldFrame = new TextFieldFrame();
10 textFieldFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11 textFieldFrame.setSize(350, 100); // configura tamanho do quadro
12 textFieldFrame.setVisible(true); // exhibe o quadro
13 } // fim de main
14 } // fim da classe TextFieldTest

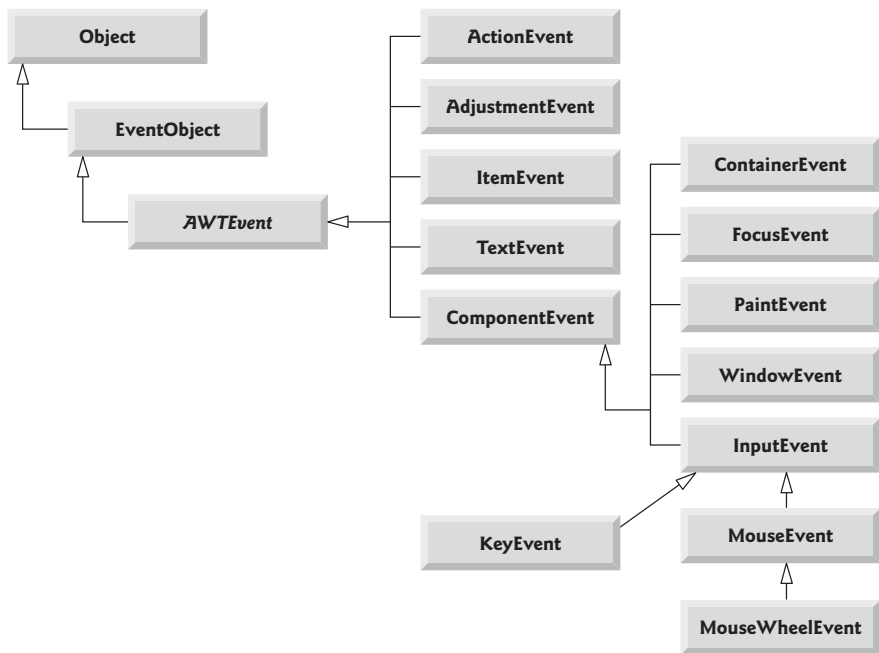
```



**Figura I.2** Testando TextFieldFrame.

## I.4 Tipos de evento comuns em interfaces gráficas e interfaces receptoras

Na seção I.3, você aprendeu que as informações sobre o evento ocorrido quando o usuário pressiona *Enter* em um campo de texto são armazenadas em um objeto `ActionEvent`. Muitos tipos de eventos diferentes podem ocorrer quando o usuário interage com uma interface gráfica. As informações sobre o evento são armazenadas em um objeto de uma classe que estende `AwtEvent` (do pacote `java.awt`). A Figura I.3 ilustra uma hierarquia contendo muitas classes de evento do pacote `java.awt.event`. Tipos de evento adicionais são declarados no pacote `javax.swing.event`.



**Figura I.3** Algumas classes de evento do pacote `java.awt.event`.

Vamos resumir as três partes do mecanismo de tratamento de evento que vimos na seção I.3 – a *origem do evento*, o *objeto evento* e o *receptor de evento*. A origem do evento é o componente da interface gráfica com a qual o usuário interage. O objeto evento encapsula informações sobre o evento ocorrido, como uma referência para a origem do evento e quaisquer dados específicos que possam ser exigidos pelo receptor de evento para tratá-lo. O receptor de evento é um objeto que é notificado pela origem do evento quando um evento ocorre; na verdade, ele “recebe” um evento, e um de seus métodos é executado em resposta ao evento. Um método do receptor de evento recebe um objeto evento quando o receptor é notificado sobre o evento. Então, o receptor utiliza o objeto evento para responder ao evento. Esse modelo de tratamento de evento é conhecido como **modelo de delegação de evento** – o processamento de um evento é delegado para um objeto (o receptor de evento) no aplicativo.

Normalmente, para cada tipo de objeto evento existe uma interface receptora de evento correspondente. Um receptor de evento de interface gráfica do usuário é um objeto de uma classe que implemente uma ou mais das interfaces receptoras de evento.

Cada interface receptora de evento especifica um ou mais métodos de tratamento de evento que *devem* ser declarados na classe que implementa a interface. Lembre-se, da seção G.12, de que qualquer classe que implemente uma interface deve declarar *todos* os métodos abstract dessa interface; caso contrário, a classe não será abstract e não poderá ser usada para criar objetos.

Quando um evento ocorre, o componente da interface gráfica com que o usuário interagiu notifica a seus *receptores registrados*, chamando o *método de tratamento de evento* apropriado de cada receptor. Por exemplo, quando o usuário pressiona a tecla *Enter* em um componente `TextField`, é chamado o método `actionPerformed` do receptor registrado. Como a rotina de tratamento de evento é registrada? Como o componente

da interface gráfica do usuário sabe chamar `actionPerformed`, em vez de outro método de tratamento de evento? Respondemos a essas perguntas e diagramamos a interação na próxima seção.

## I.5 Como funciona o tratamento de eventos

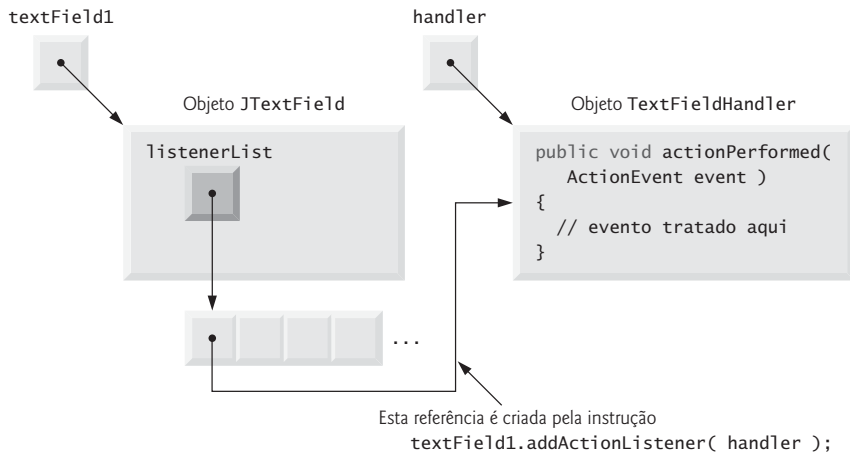
Vamos ilustrar o funcionamento do mecanismo do tratamento de eventos usando `textField1` do exemplo da Fig. I.1. Temos duas perguntas em aberto, da seção I.4:

1. Como a *rotina de tratamento de evento é registrada*?
2. Como o componente da interface gráfica do usuário sabe chamar `actionPerformed`, em vez de algum outro método de tratamento de evento?

A primeira pergunta é respondida pelo registro de evento feito nas linhas 43 a 46 da Fig. I.1. A Figura I.4 representa graficamente a variável `TextField textField1`, a variável `TextFieldHandler handler` e os objetos aos quais elas se referem.

### Registrando eventos

Todo `JComponent` tem uma variável de instância chamada `listenerList` que faz referência a um objeto da classe `EventListenerList` (pacote `javax.swing.event`). Cada objeto de uma subclasse de `JComponent` mantém referências para seus receptores registrados na variável `listenerList`. Por simplicidade, diagramamos `listenerList` como um array sob o objeto `TextField` na Fig. I.4.



**Figura I.4** Registro de evento para `textField1` de `JTextField`.

Quando a linha 43 da Fig. I.1

```
textField1.addActionListener(handler);
```

é executada, uma nova entrada, contendo uma referência para o objeto `TextFieldHandler`, é colocada em `listenerList` de `textField1`. Embora não apareça no diagrama, essa nova entrada inclui também o tipo do receptor (neste caso, `ActionListener`). Usando esse mecanismo, cada componente leve da interface gráfica do usuário Swing mantém sua própria lista de *receptores* que foram *registrados* para *tratar os eventos* do componente.



### Chamada de rotina de tratamento de evento

O tipo de receptor de evento é importante para responder à segunda pergunta: como o componente da interface gráfica do usuário sabe chamar `actionPerformed`, em vez de outro método? Todo componente de interface gráfica do usuário aceita vários *tipos de evento*, incluindo **eventos de mouse**, **eventos de tecla** e outros. Quando ocorre um evento, ele é **despachado** somente para os *receptores de evento* do tipo apropriado. Despachar é simplesmente o processo por meio do qual o componente da interface gráfica do usuário chama um método de tratamento de evento em cada um de seus receptores registrados para o tipo de evento ocorrido.

Cada *tipo de evento* tem uma ou mais *interfaces receptoras de evento* correspondentes. Por exemplo, eventos `ActionEvent` são tratados por receptoras `ActionListener`, eventos `MouseEvent` por receptoras `MouseListener` e `MouseMotionListener`, e eventos `KeyEvent` por receptoras `KeyListener`. Quando ocorre um evento, o componente da interface gráfica do usuário recebe (da JVM) uma **identificação de evento** exclusiva, especificando o tipo do evento. O componente da interface gráfica do usuário utiliza a identificação do evento para decidir sobre o tipo de receptor para o qual o evento deve ser despachado e sobre qual método deve chamar em cada objeto receptor. Para `ActionEvent`, o evento é despachado para *todo* método `actionPerformed` registrado de `ActionListener` (o único método da interface `ActionListener`). Para `MouseEvent`, o evento é despachado para *todo* receptor `MouseListener` ou `MouseMotionListener` registrado, dependendo do evento de mouse ocorrido. A identificação do evento `MouseEvent` determina quais dos vários métodos de tratamento de evento de mouse são chamados. Os componentes da interface gráfica do usuário tomam todas essas decisões para você. Basta registrar uma rotina de tratamento para o tipo de evento em particular exigido por seu aplicativo, e o componente da interface gráfica garantirá que o método apropriado da rotina de tratamento vai ser chamado quando o evento ocorrer. Discutimos outros tipos de evento e interfaces receptoras de evento quando forem necessárias, à medida que novos componentes forem apresentados.

## I.6 JButton

Um **botão** é um componente em que o usuário clica para disparar uma ação específica. Um aplicativo Java pode usar vários tipos de botões, incluindo **botões de comando**, **caixas de seleção**, **botões de alternância** e **botões de opção**. A Figura I.5 mostra a hierarquia de herança dos botões Swing abordados neste apêndice. Como você pode ver, todos os tipos de botão são subclasses de `AbstractButton` (pacote `javax.swing`), que declara os recursos comuns dos botões Swing. Nesta seção, nos concentramos nos botões que normalmente são usados para iniciar um comando.

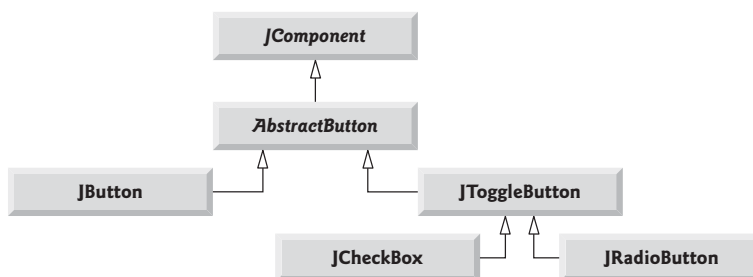


Figura I.5 Hierarquia de botões Swing.

Um botão de comando (consulte a saída da Fig. I.7) gera um evento `ActionEvent` quando o usuário clica nele. Os botões de comando são criados com a classe `JButton`. O texto da face de um componente `JButton` é chamado de **rótulo do botão**. Uma interface gráfica do usuário pode ter muitos componentes `JButton`, mas cada rótulo deve ser exclusivo na parte da interface exibida no momento.



#### Observação sobre aparência e comportamento I.1

*Normalmente, o texto nos botões utilizam letras maiúsculas e minúsculas como no título de um livro.*



#### Observação sobre aparência e comportamento I.2

*Ter mais de um componente `JButton` com o mesmo rótulo os torna ambíguos para o usuário. Forneça um rótulo exclusivo para cada botão.*

O aplicativo das Figs. I.6 e I.7 cria dois componentes `JButton` e demonstra que eles suportam a exibição de elementos `Icon`. O tratamento de evento para os botões é realizado por uma única instância da *classe interna* `ButtonHandler` (linhas 39 a 47).

```

1 // Fig. I.6: ButtonFrame.java
2 // Botões de comando e eventos de ação.
3 import java.awt.FlowLayout;
4 import java.awt.event.ActionListener;
5 import java.awt.event.ActionEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JButton;
8 import javax.swing.Icon;
9 import javax.swing.ImageIcon;
10 import javax.swing.JOptionPane;
11
12 public class ButtonFrame extends JFrame
13 {
14 private JButton plainJButton; // botão apenas com texto
15 private JButton fancyJButton; // botão com ícones
16
17 // ButtonFrame adiciona componentes JButton a JFrame
18 public ButtonFrame()
19 {
20 super("Testing Buttons");
21 setLayout(new FlowLayout()); // configura o layout do quadro
22
23 plainJButton = new JButton("Plain Button"); // botão com texto
24 add(plainJButton); // adiciona plainJButton a JFrame
25
26 Icon bug1 = new ImageIcon(getClass().getResource("bug1.gif"));
27 Icon bug2 = new ImageIcon(getClass().getResource("bug2.gif"));
28 fancyJButton = new JButton("Fancy Button", bug1); // configura imagem
29 fancyJButton.setRolloverIcon(bug2); // configura imagem de transição
30 add(fancyJButton); // adiciona fancyJButton a JFrame
31
32 // cria novo ButtonHandler para o tratamento de evento de botão
33 ButtonHandler handler = new ButtonHandler();
34 fancyJButton.addActionListener(handler);
35 plainJButton.addActionListener(handler);

```

**Figura I.6** Botões de comando e eventos de ação. (continua)

```

36 } // fim do construtor de ButtonFrame
37
38 // classe interna para tratamento de eventos de botão
39 private class ButtonHandler implements ActionListener
40 {
41 // trata eventos de botão
42 public void actionPerformed((ActionEvent event))
43 {
44 JOptionPane.showMessageDialog(ButtonFrame.this, String.format(
45 "You pressed: %s", event.getActionCommand()));
46 } // fim do método actionPerformed
47 } // fim da classe interna private ButtonHandler
48 } // fim da classe ButtonFrame

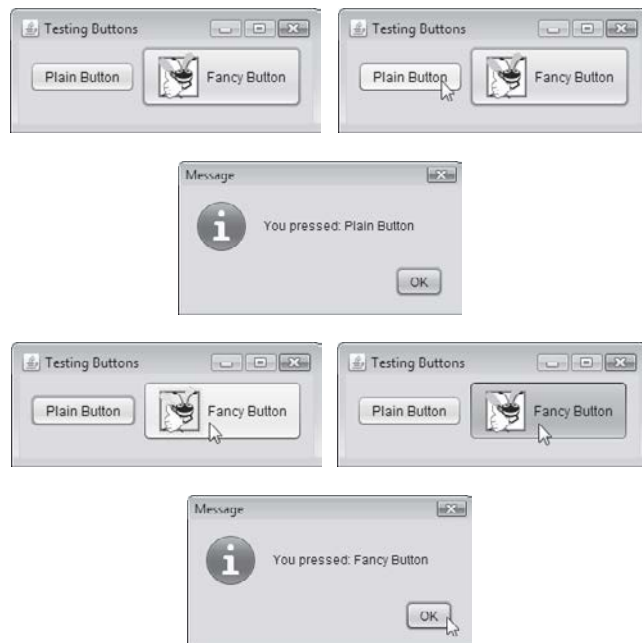
```

**Figura I.6** Botões de comando e eventos de ação.

```

1 // Fig. I.7: ButtonTest.java
2 // Testando ButtonFrame.
3 import javax.swing.JFrame;
4
5 public class ButtonTest
6 {
7 public static void main(String[] args)
8 {
9 ButtonFrame buttonFrame = new ButtonFrame(); // cria ButtonFrame
10 buttonFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11 buttonFrame.setSize(275, 110); // configura tamanho do quadro
12 buttonFrame.setVisible(true); // exibe o quadro
13 } // fim de main
14 } // fim da classe ButtonTest

```



**Figura I.7** Testando ButtonFrame.

As linhas 14 e 15 da Fig. I.6 declaram as variáveis `JButton plainJButton` e `fancyJButton`. Os objetos correspondentes são instanciados no construtor. A linha 23 cria `plainJButton` com o rótulo "Plain Button". A linha 24 adiciona o `JButton` a `JFrame`.

Um `JButton` pode exibir um elemento `Icon` (ícone). Para fornecer ao usuário um nível extra de interação visual com a interface gráfica, um componente `JButton` também pode ter um **ícone (Icon) de transição** – um elemento `Icon` exibido quando o usuário posiciona o mouse sobre o componente `JButton`. O ícone do `JButton` muda à medida que o mouse entre e sai da área do componente `JButton` na tela. As linhas 26 e 27 (Fig. I.6) criam dois objetos `ImageIcon` que representam o elemento `Icon` padrão e o de transição para o componente `JButton` criado na linha 28. As duas instruções presumem que os arquivos de imagem estão armazenados no mesmo diretório do aplicativo. Imagens são comumente colocadas no mesmo diretório do aplicativo ou em um subdiretório, como `images`. Esses arquivos de imagem foram fornecidos com o exemplo para você.

A linha 28 cria `fancyButton` com o texto "Fancy Button" e o ícone `bug1`. Por padrão, o texto é exibido à direita do ícone. A linha 29 usa `setRolloverIcon` (herdado da classe `AbstractButton`) para especificar a imagem exibida no componente `JButton` quando o usuário posiciona o mouse sobre ele. A linha 30 adiciona o `JButton` a `JFrame`.



### Observação sobre aparência e comportamento I.3

*Como a classe `AbstractButton` aceita a exibição de texto e imagens em um botão, todas as suas subclasses também aceitam.*



### Observação sobre aparência e comportamento I.4

*O uso de ícones de transição para componentes `JButton` proporciona ao usuário um retorno visual indicando que, quando ele clicar com o mouse enquanto o cursor estiver posicionado sobre o componente `JButton`, ocorrerá uma ação.*

Como os componentes `JTextField`, os componentes `JButton` geram eventos `ActionEvent` que podem ser processados por qualquer objeto `ActionListener`. As linhas 33 a 35 criam um objeto da classe interna `private ButtonHandler` e utilizam `addActionListener` para registrá-lo como rotina de tratamento de evento para cada componente `JButton`. A classe `ButtonHandler` (linhas 39 a 47) declara `actionPerformed` para exibir uma caixa de diálogo de mensagem contendo o rótulo para o botão pressionado pelo usuário. Para um evento de `JButton`, o método `getActionCommand` de `ActionEvent` retorna o rótulo do `JButton`.

### Acessando a referência `this` em um objeto de uma classe de nível superior a partir de uma classe aninhada

Quando você executar este aplicativo e clicar em um de seus botões, observe que a caixa de diálogo de mensagem que aparece fica centralizada sobre a janela do aplicativo. Isso ocorre porque a chamada ao método `showMessageDialog` de `JOptionPane` (linhas 44 e 45 da Fig. I.6) utiliza `ButtonFrame.this`, em vez de `null`, como primeiro argumento. Quando esse argumento não é `null`, ele representa o assim chamado *componente da interface gráfica de usuário pai* da caixa de diálogo de mensagem (neste caso, a janela do aplicativo é o componente pai) e permite que, ao ser exibida, a caixa de diálogo seja centralizada sobre esse componente. `ButtonFrame.this` representa a referência `this` do objeto da classe de nível superior `ButtonFrame`.



### Observação sobre engenharia de software I.2

*Quando usada em uma classe interna, a palavra-chave `this` se refere ao objeto de classe interna que está sendo manipulado no momento. Um método de classe interna pode usar a referência `this` do objeto de sua classe externa precedendo-a com o nome da classe externa e um ponto, como em `ButtonFrame.this`.*

## I.7 JComboBox: uso de uma classe interna anônima para tratamento de eventos

Uma caixa de combinação (às vezes chamada de **lista suspensa**) permite ao usuário selecionar um item de uma lista (Fig. I.9). As caixas de combinação são implementadas com a classe `JComboBox`, a qual estende a classe `JComponent`. Os componentes `JComboBox` geram eventos `ItemEvent`, exatamente como acontece com os componentes `JCheckBox` e `JRadioButton`. Este exemplo demonstra também uma forma especial de classe interna, frequentemente utilizada em tratamento de evento. O aplicativo (Figs. I.8 e I.9) usa um componente `JComboBox` para fornecer uma lista de quatro nomes de arquivos de imagem, nos quais o usuário pode selecionar uma imagem para exibição. Quando o usuário seleciona um nome, o aplicativo exibe a imagem correspondente como um elemento `Icon` em um componente `JLabel`. A classe `ComboBoxTest` (Fig. I.9) contém o método `main` que executa este aplicativo. As capturas de tela para este aplicativo mostram a lista de `JComboBox` após ser feita a seleção para ilustrar qual nome de arquivo de imagem foi selecionado.

As linhas 19 a 23 (Fig. I.8) declaram e inicializam o array `icons` com quatro novos objetos `ImageIcon`. O array de `String` `names` (linhas 17 e 18) contém os nomes dos quatro arquivos de imagem armazenados no mesmo diretório do aplicativo.

```

1 // Fig. I.8: ComboBoxFrame.java
2 // JComboBox que exibe uma lista de nomes de imagem.
3 import java.awt.FlowLayout;
4 import java.awt.event.ItemListener;
5 import java.awt.event.ItemEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JLabel;
8 import javax.swing.JComboBox;
9 import javax.swing.Icon;
10 import javax.swing.ImageIcon;
11
12 public class ComboBoxFrame extends JFrame
13 {
14 private JComboBox imagesJComboBox; // caixa de combinação para
15 // armazenar nomes de ícones
16 private JLabel label; // rótulo para exibir o ícone selecionado
17
18 private static final String[] names =
19 { "bug1.gif", "bug2.gif", "travelbug.gif", "buganim.gif" };
20 private Icon[] icons = {
21 new ImageIcon(getClass().getResource(names[0])),
22 new ImageIcon(getClass().getResource(names[1])),
23 new ImageIcon(getClass().getResource(names[2])),
24 new ImageIcon(getClass().getResource(names[3])) };
25
26 // o construtor de ComboBoxFrame adiciona JComboBox a JFrame
27 public ComboBoxFrame()

```

**Figura I.8** JComboBox que exibe uma lista de nomes de imagem. (continua)

```

27 {
28 super("Testing JComboBox");
29 setLayout(new FlowLayout()); // configura o layout do quadro
30
31 imagesJComboBox = new JComboBox(names); // configura JComboBox
32 imagesJComboBox.setMaximumRowCount(3); // exibe três linhas
33
34 imagesJComboBox.addItemListener(
35 new ItemListener() // classe interna anônima
36 {
37 // trata evento de JComboBox
38 public void itemStateChanged(ItemEvent event)
39 {
40 // determina se o item foi selecionado
41 if (event.getStateChange() == ItemEvent.SELECTED)
42 label.setIcon(icons[
43 imagesJComboBox.getSelectedIndex()]);
44 } // fim do método itemStateChanged
45 } // fim da classe interna anônima
46); // fim da chamada a addItemListener
47
48 add(imagesJComboBox); // adiciona caixa de combinação a JFrame
49 label = new JLabel(icons[0]); // exibe o primeiro ícone
50 add(label); // adiciona rótulo a JFrame
51 } // fim do construtor de ComboBoxFrame
52 } // fim da classe ComboBoxFrame

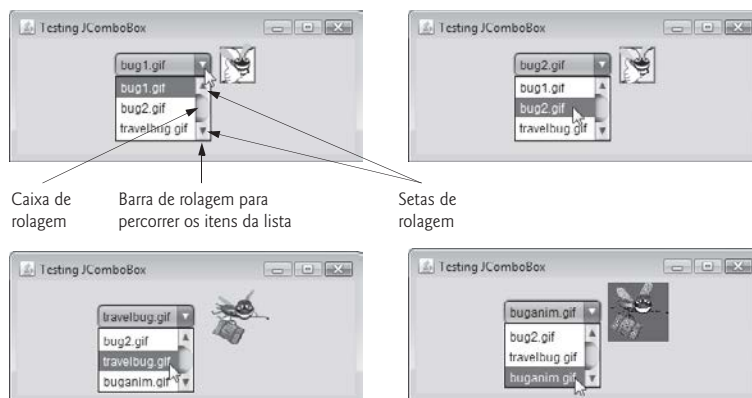
```

**Figura I.8** JComboBox que exibe uma lista de nomes de imagem.

```

1 // Fig. I.9: ComboBoxTest.java
2 // Testando ComboBoxFrame.
3 import javax.swing.JFrame;
4
5 public class ComboBoxTest
6 {
7 public static void main(String[] args)
8 {
9 ComboBoxFrame comboBoxFrame = new ComboBoxFrame();
10 comboBoxFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11 comboBoxFrame.setSize(350, 150); // configura tamanho do quadro
12 comboBoxFrame.setVisible(true); // exibe o quadro
13 } // fim de main
14 } // fim da classe ComboBoxTest

```



**Figura I.9** Testando ComboBoxFrame.

Na linha 31, o construtor inicializa um objeto `JComboBox` com as Strings do array `names` como elementos da lista. Cada item da lista tem um **índice**. O primeiro item é adicionado ao índice 0, o seguinte ao índice 1 e assim por diante. O primeiro item adicionado a um componente `JComboBox` aparece como item atualmente selecionado quando o `JComboBox` é exibido. Outros itens são selecionados clicando no componente `JComboBox` e, então, selecionando um item na lista que aparece.

A linha 32 usa o método `setMaximumRowCount` de `JComboBox` para configurar o número máximo de elementos que são exibidos quando o usuário clica no componente `JComboBox`. Se houver mais itens, o componente `JComboBox` fornecerá uma **barra de rolagem** (consulte a primeira tela), a qual permite ao usuário percorrer todos os elementos da lista. O usuário pode clicar nas **setas de rolagem**, na parte superior e inferior da barra de rolagem, para subir ou descer na lista um elemento por vez, ou então arrastar para cima ou para baixo a **caixa de rolagem** no meio da barra de rolagem. Para arrastar a caixa de rolagem, posicione o cursor do mouse sobre ela, mantenha o botão do mouse pressionado e mova o mouse. Neste exemplo, a lista suspensa é curta demais para arrastar a caixa de rolagem, de modo que você pode clicar nas setas para cima e para baixo ou usar a roda do mouse para rolar pelos quatro itens da lista.



#### Observação sobre aparência e comportamento I.5

*Configure a contagem de linhas máxima de um componente `JComboBox` com um número de linhas que impeça a expansão da lista para fora dos limites da janela na qual ela é usada.*

A linha 48 anexa o componente `JComboBox` ao elemento `FlowLayout` (configurado na linha 29) de `ComboBoxFrame`. A linha 49 cria o componente `JLabel` que exibe elemento `ImageIcon` e os inicializa com o primeiro elemento `ImageIcon` do array `icons`. A linha 50 anexa o componente `JLabel` ao elemento `FlowLayout` de `ComboBoxFrame`.

#### Usando uma classe interna anônima para tratamento de eventos

As linhas 34 a 46 compõem uma única instrução que declara a classe do receptor de evento, cria um objeto dessa classe e o registra como receptor para eventos `ItemEvent` de `imagesJComboBox`. Esse objeto receptor de evento é uma instância de uma **classe interna anônima** – uma classe interna declarada sem um nome e que normalmente aparece dentro de uma declaração de método. *Como as outras classes internas, uma classe interna anônima pode acessar os membros de sua classe de nível superior.* No entanto, uma classe interna anônima tem acesso limitado às variáveis locais do método no qual é declarada. Visto que uma classe interna anônima não tem nome, um objeto da classe deve ser criado no ponto onde ela é declarada (a partir da linha 35).



#### Observação sobre engenharia de software I.3

*Uma classe interna anônima declarada em um método pode acessar as variáveis de instância e os métodos do objeto de classe de nível superior que a declarou, assim como as variáveis locais `final` do método, mas não pode acessar suas variáveis locais não `final`.*

As linhas 34 a 46 chamam o método `addItemListener` de `imagesJComboBox`. O argumento desse método deve ser um objeto `ItemListener` (isto é, qualquer objeto de uma classe que implemente `ItemListener`). As linhas 35 a 45 são uma expressão de criação de instância de classe que declara uma classe interna anônima e cria um objeto dessa classe. Então, uma referência para esse objeto é passada como argumento para

addItemListener. A sintaxe ItemListener() após new inicia a declaração de uma classe interna anônima que implementa a interface ItemListener. Isso é semelhante a iniciar uma declaração de classe com

```
public class MyHandler implements ItemListener
```

A chave de abertura na linha 36 e a de fechamento na linha 45 delimitam o corpo da classe interna anônima. As linhas 38 a 44 declaram o método itemStateChanged de ItemListener. Quando o usuário faz uma seleção em imagesJComboBox, esse método configura o elemento Icon do rótulo. O elemento Icon é selecionado do array icons pela determinação do índice do item selecionado no componente JComboBox com o método `getSelectedIndex`, na linha 43. Para cada item selecionado de um componente JComboBox, primeiramente outro item perde a seleção – portanto, ocorrem dois eventos ItemEvent quando um item é selecionado. Queremos exibir apenas o ícone do item que o usuário acabou de selecionar. Por isso, a linha 41 determina se o método `getStateChange` de ItemEvent retorna ItemEvent.SELECTED. Em caso positivo, as linhas 42 e 43 configuram o ícone do rótulo.



**Observação sobre engenharia de software I.4**

*Como em qualquer outra classe, quando uma classe interna anônima implementa uma interface, deve implementar todos os métodos da interface.*

A sintaxe mostrada nas linhas 35 a 45 para criar uma rotina de tratamento de evento com uma classe interna anônima é semelhante ao código que seria gerado por um IDE Java. Normalmente, um IDE permite projetar uma interface gráfica do usuário visualmente e, então, gera código que implementa a interface. Basta inserir, nos métodos de tratamento de evento, instruções que declarem como tratar cada evento.

I.8 Classes adaptadoras

Muitas interfaces receptoras de evento, como MouseListener e MouseMotionListener, contêm vários métodos. Nem sempre se quer declarar cada método de uma interface receptora de evento. Por exemplo, talvez um aplicativo precise apenas da rotina de tratamento mouseClicked de MouseListener ou da rotina de tratamento mouseDragged de MouseMotionListener. A interface WindowListener especifica sete métodos de tratamento de evento de janela. Para muitas das interfaces receptoras que possuem vários métodos, os pacotes java.awt.event e javax.swing.event fornecem classes adaptadoras e receptoras de evento. Uma **classe adaptadora** implementa uma interface e fornece uma implementação padrão (com um corpo de método vazio) de cada método da interface. É possível estender uma classe adaptadora para herdar a implementação padrão de cada método e, subsequentemente, sobrescrever apenas o método (ou métodos) necessário para tratamento de evento.



**Observação sobre engenharia de software I.5**

*Quando uma classe implementa uma interface, a classe tem uma relação é um com essa interface. Todas as subclasses diretas e indiretas dessa classe herdam essa interface. Assim, um objeto de uma classe que estende uma classe adaptadora de evento é um objeto do tipo receptor de evento correspondente (por exemplo, um objeto de uma subclasse de MouseAdapter é um MouseListener).*



## I.9 Para finalizar

Neste apêndice, você aprendeu sobre alguns componentes de interface gráfica do usuário e como implementar rotinas de tratamento de evento utilizando classes aninhadas e classes internas anônimas. Viu a relação especial entre um objeto de classe interna e um objeto de sua classe de nível superior. Você também aprendeu a criar aplicativos que são executados em suas próprias janelas. Discutimos a classe `JFrame` e os componentes que permitem a um usuário interagir com um aplicativo.

### Exercícios de revisão

- I.1** Preencha os espaços em branco em cada um dos seguintes enunciados:
- Um(a) \_\_\_\_\_ organiza os componentes da interface gráfica do usuário em um objeto `Container`.
  - O método `add`, usado para anexar componentes de interface gráfica do usuário, é um método da classe \_\_\_\_\_.
  - GUI é o acrônimo de \_\_\_\_\_.
  - O método \_\_\_\_\_ é usado para especificar o gerenciador de layout para um contêiner.
- I.2** Determine se a declaração a seguir é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo: Classes internas não podem acessar os membros da classe circundante.

### Respostas dos exercícios de revisão

- I.1** a) gerenciador de layout. b) `Container`. c) interface gráfica do usuário. d) `setLayout`.
- I.2** Falsa. As classes internas têm acesso a todos os membros da declaração de classe circundante.

### Exercícios

- I.3** (*Conversão de temperatura*) Escreva um aplicativo de conversão de temperatura que converta de Fahrenheit para Celsius. A temperatura em Fahrenheit deve ser inserida a partir do teclado (por meio de um componente `TextField`). Deve ser usado um componente `JLabel` para mostrar a temperatura convertida. Use a seguinte fórmula para a conversão:

$$Celsius = \frac{5}{9} \times (Fahrenheit - 32)$$

- I.4** (*Modificação do aplicativo de conversão de temperatura*) Aprimore o aplicativo de conversão de temperatura do Exercício I.3, adicionando a escala de temperaturas Kelvin. O aplicativo também deve permitir que o usuário faça conversões entre quaisquer duas escalas. Use a fórmula a seguir para a conversão entre Kelvin e Celsius (além da fórmula do Exercício I.3):

$$Kelvin = Celsius + 273.15$$

- I.5** (*Jogo “adivinhe o número”*) Escreva um aplicativo que jogue “adivinhar o número”, como segue: seu aplicativo escolhe o número a ser adivinhado selecionando um valor inteiro aleatório no intervalo de 1 a 1000. Então, o aplicativo exibe o seguinte em um rótulo:

Tenho um número entre 1 e 1000. Consegue adivinhar meu número?  
Por favor, digite sua primeira tentativa:

Um componente `TextField` deve ser usado para inserir a tentativa. À medida que cada tentativa é inserida, a cor de fundo muda para vermelho ou azul. Vermelho indica que o usuário está “esquentando” e azul, “esfriando”. Um elemento `Label` deve exibir “Alto demais” ou “Baixo demais” para ajudar o usuário a fechar o cerco. Quando o usuário acerta a resposta, deve ser exibido “Correto!”, e o componente `TextField` utilizado para entrada deve mudar para não editável. Deve ser fornecido um componente `Button` para permitir ao usuário jogar novamente. Quando o componente `Button` é clicado, deve ser gerado um novo número aleatório, e o componente `TextField` de entrada deve ser alterado para editável.

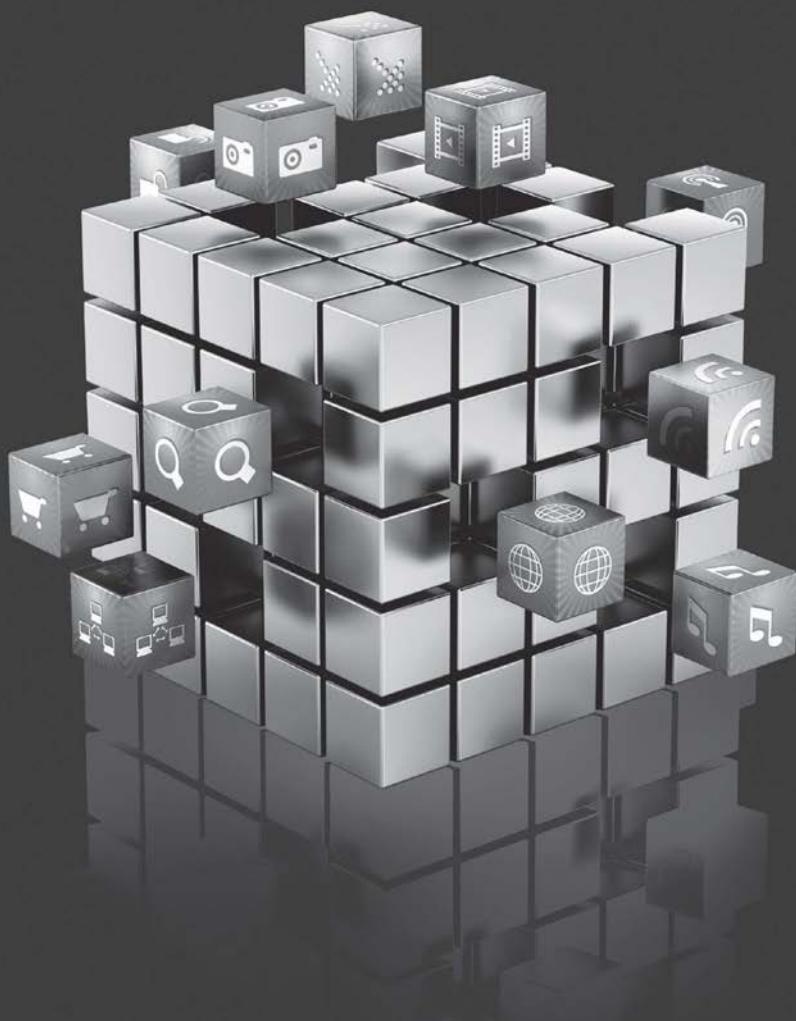
# J

## Outros tópicos da linguagem Java

### Objetivos

Neste capítulo, você vai:

- Aprender o que são coleções.
- Usar a classe `Arrays` para manipulações de array.
- Entender como as classes encapsuladoras de tipos possibilitam aos programas processar valores de dados primitivos como objetos.
- Usar estruturas de dados genéricas prontas do framework de coleções.
- Usar iteradores para “percorrer” uma coleção.
- Aprender conceitos fundamentais sobre processamento de arquivos e de fluxos.
- Saber o que são threads e porque elas são úteis.
- Saber como as threads permitem gerenciar atividades concomitantes.
- Criar e executar interfaces `Runnable`.
- Conhecer os fundamentos da sincronização de threads.
- Saber como várias threads podem atualizar componentes de interface gráfica do usuário Swing de modo seguro.



## Resumo

|       |                                                                                     |      |                                                                     |
|-------|-------------------------------------------------------------------------------------|------|---------------------------------------------------------------------|
| J.1   | Introdução                                                                          | J.8  | Conjuntos                                                           |
| J.2   | Visão geral das coleções                                                            | J.9  | Mapas                                                               |
| J.3   | Classes encapsuladoras de tipos primitivos                                          | J.10 | Introdução aos arquivos e fluxos                                    |
| J.4   | Interface <code>Collection</code> e classe <code>Collections</code>                 | J.11 | Classe <code>File</code>                                            |
| J.5   | Listas                                                                              | J.12 | Introdução à serialização de objetos                                |
| J.5.1 | <code>ArrayList</code> e <code>Iterator</code>                                      | J.13 | Introdução ao multithread                                           |
| J.5.2 | <code>LinkedList</code>                                                             | J.14 | Criação e execução de threads com o framework <code>Executor</code> |
| J.5.3 | Modos de exibição em coleções e o método <code>asList</code> de <code>Arrays</code> | J.15 | Visão geral da sincronização de threads                             |
| J.6   | Métodos de <code>Collections</code>                                                 | J.16 | Visão geral das coleções concorrentes                               |
| J.6.1 | Método <code>sort</code>                                                            | J.17 | Multithread com interface gráfica do usuário                        |
| J.6.2 | Método <code>shuffle</code>                                                         | J.18 | Para finalizar                                                      |
| J.7   | Interface <code>Queue</code>                                                        |      |                                                                     |

*Exercícios de revisão | Respostas dos exercícios de revisão | Exercícios*

## J.1 Introdução

Este apêndice apresenta vários tópicos adicionais em apoio à parte sobre Android do livro. As seções J.2 a J.9 apresentam uma visão geral do framework de coleções do Java e vários exemplos de como trabalhar com diversas coleções que utilizamos em nossos aplicativos Android. As seções J.10 a J.12 apresentam conceitos sobre arquivos e fluxos, examinam o método da classe `File` e discutem a serialização de objetos para escrever objetos inteiros em fluxos e ler objetos inteiros de fluxos. Por último, as seções J.13 a J.17 apresentam os fundamentos do multithread.

## J.2 Visão geral das coleções

A seção E.12 apresentou a coleção genérica `ArrayList` – uma estrutura de dados do tipo array que pode ser redimensionada e que armazena referências para objetos de um tipo especificado por você ao criar a `ArrayList`. Agora, continuamos nossa discussão sobre o **framework de coleções** Java, o qual contém muitas outras estruturas de dados genéricas predefinidas e vários métodos para manipulá-las. Nos concentramos nas que são utilizadas nos capítulos sobre Android deste livro e nas que têm forte semelhança nas APIs Android. Para ver os detalhes completos do framework de coleções, visite

[docs.oracle.com/javase/6/docs/technotes/guides/collections/](https://docs.oracle.com/javase/6/docs/technotes/guides/collections/)

Uma **coleção** é uma estrutura de dados – na verdade, um objeto – que pode armazenar referências para outros objetos. Normalmente, as coleções contêm referências para objetos que são todos do mesmo tipo. As interfaces do framework de coleções declaram as operações a serem efetuadas genericamente em vários tipos de coleções. A Figura J.1 lista algumas das interfaces do framework de coleções. Várias implementações dessas interfaces são fornecidas dentro do framework. Você também pode fornecer implementações específicas aos seus próprios requisitos.

Como o tipo a armazenar em uma coleção é especificado no momento da compilação, as coleções genéricas oferecem segurança quanto ao tipo em tempo de compilação, o que permite ao compilador capturar tentativas de usar tipos inválidos. Por exemplo, não é possível armazenar objetos `Employee` em uma coleção de objetos `String`.

| Interface               | Descrição                                                                                                                                                                  |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Collection</code> | A interface-raiz da hierarquia de coleções, a partir da qual são derivadas as interfaces <code>Set</code> , <code>Queue</code> e <code>List</code> .                       |
| <code>Set</code>        | Coleção que não contém duplicatas.                                                                                                                                         |
| <code>List</code>       | Coleção ordenada que pode conter elementos duplicados.                                                                                                                     |
| <code>Map</code>        | Coleção que associa chaves a valores e não pode conter chaves duplicadas.                                                                                                  |
| <code>Queue</code>      | Normalmente, uma coleção do tipo “primeiro a entrar, primeiro a sair” ( <i>first-in, first-out</i> ) que modela uma fila de espera; outras ordens podem ser especificadas. |

**Figura J.1** Algumas interfaces do framework de coleções.

Alguns exemplos de coleções são as cartas que você tem em mãos em um jogo de baralho, suas músicas prediletas armazenadas em seu computador, os membros de uma equipe esportiva e os registros de bens imobiliários no cartório de sua localidade (que fazem o mapeamento dos números da escritura e da página para os proprietários de imóveis).

### J.3 Classes encapsuladoras de tipos primitivos

Cada tipo primitivo (listado no Apêndice L) tem uma **classe encapsuladora de tipos** correspondente no pacote `java.lang`. São elas: **`Boolean`**, **`Byte`**, **`Character`**, **`Double`**, **`Float`**, **`Integer`**, **`Long`** e **`Short`**. Elas permitem manipular valores de tipo primitivo como objetos. As estruturas de dados reutilizáveis da linguagem Java manipulam e compartilham *objetos* – elas não podem manipular variáveis de tipos primitivos. Contudo, podem manipular objetos das classes encapsuladoras de tipos, pois, em última análise, toda classe deriva de `Object`.

Cada uma das classes encapsuladoras de tipos numéricos – `Byte`, `Short`, `Integer`, `Long`, `Float` e `Double` – estende a classe `Number`. Além disso, as classes wrapper de tipo são final; portanto, não é possível estendê-las.

Os tipos primitivos não têm métodos, de modo que os métodos relacionados a um tipo primitivo estão localizados na classe encapsuladora de tipos correspondente (por exemplo, o método `parseInt`, que converte uma `String` e um valor `int`, está localizado na classe `Integer`). Caso você precise manipular um valor primitivo em seu programa, consulte primeiramente a documentação das classes encapsuladoras de tipos – talvez o método necessário já esteja declarado.

#### **Autoboxing e auto-unboxing**

A linguagem Java fornece *conversões boxing* (encaixotamento) e *unboxing* (desencaixotamento) para converter automaticamente entre valores de tipo primitivo e objetos encapsuladores de tipos. A **conversão boxing** converte um valor de tipo primitivo em um objeto da classe encapsuladora de tipos correspondente. A **conversão unboxing** converte um objeto de uma classe encapsuladora de tipos para um valor de tipo primitivo correspondente. Essas conversões são feitas automaticamente (são chamadas de **autoboxing** e **auto-unboxing**), permitindo que valores de tipo primitivo sejam usados onde são esperados objetos encapsuladores de tipos e vice-versa.

### J.4 Interface `Collection` e classe `Collections`

A interface `Collection` é a raiz da hierarquia de coleções, a partir da qual são derivadas as interfaces `Set`, `Queue` e `List`. A interface `Set` define uma coleção que não contém duplicatas. A interface `Queue` define uma coleção que representa uma fila de espera –

normalmente, as inserções são feitas no final de uma fila e as exclusões a partir do início, embora outras ordens possam ser especificadas. Discutimos `Queue` e `Set` nas seções J.7 e J.8. A interface `Collection` contém **operações de massa** (isto é, operações efetuadas em toda uma coleção) para transações como adicionar, limpar e comparar objetos (ou elementos) em uma coleção. Um objeto `Collection` também pode ser convertido em um array. Além disso, a interface `Collection` fornece um método que retorna um objeto **Iterator**, o qual permite a um programa percorrer a coleção e remover elementos dela durante a iteração. Discutimos a classe `Iterator` na seção J.5.1. Outros métodos da interface `Collection` permitem a um programa determinar o tamanho de uma coleção e se ela está vazia. A classe `Collections` fornece métodos estáticos que pesquisam, ordenam e efetuam outras operações em coleções. A seção J.6 discute os métodos que estão disponíveis na classe `Collections`.



#### Observação sobre engenharia de software J.1

*A maioria das implementações de coleção fornece um construtor que recebe um argumento `Collection`, permitindo com isso a construção de uma nova coleção contendo os elementos da coleção especificada.*

## J.5 Listas

Um objeto `List` é uma coleção ordenada que pode conter elementos duplicados. Assim como os índices de array, os índices de `List` são baseados em zero (isto é, o índice do primeiro elemento é zero). Além dos métodos herdados de `Collection`, a interface `List` fornece métodos para manipular elementos por meio de seus índices, manipular um intervalo especificado de elementos, procurar elementos e obter um objeto `ListIterator` para acessar os elementos.

A interface `List` é implementada por várias classes, incluindo `ArrayList` (apresentada no Apêndice E) e `LinkedList`. A classe `ArrayList` é uma implementação de array dimensionável de `List`. Inserir um elemento entre outros já existentes em uma `ArrayList` é uma operação *ineficiente* – todos os elementos após o novo elemento devem mudar de lugar, operação que pode ser dispendiosa em uma coleção com um grande número de elementos. Uma `LinkedList` permite a inserção (ou remoção) eficiente de elementos no meio de uma coleção. As duas subseções a seguir demonstram vários recursos de `List` e `Collection`.

### J.5.1 ArrayList e Iterator

A Figura J.2 usa um objeto `ArrayList` (apresentado na seção E.12) para demonstrar vários recursos da interface `Collection`. O programa coloca dois arrays `Color` em objetos `ArrayList` e utiliza um objeto `Iterator` para remover do primeiro elementos da segunda coleção `ArrayList`.

```

1 // Fig. J.2: CollectionTest.java
2 // Interface Collection demonstrada por meio de um objeto ArrayList.
3 import java.util.List;
4 import java.util.ArrayList;
5 import java.util.Collection;
6 import java.util.Iterator;
7
8 public class CollectionTest

```

**Figura J.2** Interface `Collection` demonstrada por meio de um objeto `ArrayList`. (continua)

```

9 {
10 public static void main(String[] args)
11 {
12 // adiciona elementos do array colors à lista
13 String[] colors = { "MAGENTA", "RED", "WHITE", "BLUE", "CYAN" };
14 List< String > list = new ArrayList< String >();
15
16 for (String color : colors)
17 list.add(color); // adiciona cor ao final da lista
18
19 // adiciona elementos do array removeColors a removeList
20 String[] removeColors = { "RED", "WHITE", "BLUE" };
21 List< String > removeList = new ArrayList< String >();
22
23 for (String color : removeColors)
24 removeList.add(color);
25
26 // gera o conteúdo da lista na saída
27 System.out.println("ArrayList: ");
28
29 for (int count = 0; count < list.size(); count++)
30 System.out.printf("%s ", list.get(count));
31
32 // remove da lista as cores contidas em removeList
33 removeColors(list, removeList);
34
35 // gera o conteúdo da lista na saída
36 System.out.println("\n\nArrayList after calling removeColors: ");
37
38 for (String color : list)
39 System.out.printf("%s ", color);
40 } // fim de main
41
42 // remove de collection1 as cores especificadas em collection2
43 private static void removeColors(Collection< String > collection1,
44 Collection< String > collection2)
45 {
46 // obtém iterador
47 Iterator< String > iterator = collection1.iterator();
48
49 // faz loop enquanto a coleção tem itens
50 while (iterator.hasNext())
51 {
52 if (collection2.contains(iterator.next()))
53 iterator.remove(); // remove a cor atual
54 } // fim do while
55 } // fim do método removeColors
56 } // fim da classe CollectionTest

```

```

ArrayList:
MAGENTA RED WHITE BLUE CYAN

ArrayList after calling removeColors:
MAGENTA CYAN

```

**Figura J.2** Interface `Collection` demonstrada por meio de um objeto `ArrayList`.

As linhas 13 e 20 declaram e inicializam os arrays `String colors` e `removeColors`. As linhas 14 e 21 criam objetos `ArrayList<String>` e atribuem suas referências às variáveis `List<String> list` e `removeList`, respectivamente. Neste exemplo, nos referimos a objetos `ArrayList` por meio de variáveis `List`. Isso torna nosso código mais flexível e

fácil de modificar. Se, posteriormente, decidirmos que seriam mais adequados objetos `LinkedList`, precisaremos modificar somente as linhas 14 e 21, onde criamos os objetos `ArrayList`.

As linhas 16 e 17 preenchem a lista com as `Strings` armazenadas no array `colors` e as linhas 23 e 24 preenchem `removeList` com as `Strings` armazenadas no array `removeColors` usando o **método** `add` de `List`. As linhas 29 e 30 geram cada elemento da lista na saída. A linha 29 chama o **método** `size` de `List` para obter o número de elementos no objeto `ArrayList`. A linha 30 usa o método `get` de `List` para recuperar valores de elementos individuais. As linhas 29 e 30 também poderiam ter usado a instrução `for` melhorada (a qual vamos demonstrar com coleções em outros exemplos).

A linha 33 chama o método `removeColors` (linhas 43 a 55), passando `list` e `removeList` como argumentos. O método `removeColors` exclui as `Strings` que estão em `removeList` das `Strings` que estão em `list`. As linhas 38 e 39 imprimem os elementos de `list` após `removeColors` completar sua tarefa.

O método `removeColors` declara dois parâmetros `Collection<String>` (linhas 43 e 44) que permitem a quaisquer dois objetos `Collection` contendo strings serem passados como argumentos para esse método. O método acessa os elementos do primeiro objeto `Collection` (`collection1`) por meio de um objeto `Iterator`. A linha 47 chama o método `iterator` de `Collection` para obter um objeto `Iterator` para o objeto `Collection`. As interfaces `Collection` e `Iterator` são tipos genéricos. A condição de continuação de loop (linha 50) chama o método `hasNext` de `Iterator` para determinar se o objeto `Collection` contém mais elementos. O método `hasNext` retorna `true` se existe outro elemento; caso contrário, retorna `false`.

A condição `if` na linha 52 chama o **método** `next` de `Iterator` para obter uma referência para o próximo elemento e, então, usa o método `contains` do segundo objeto `Collection` (`collection2`) para determinar se `collection2` contém o elemento retornado por `next`. Em caso positivo, a linha 53 chama o **método** `remove` de `Iterator` para remover o elemento do objeto `Collection` `collection1`.



### Erro de programação comum J.1

*Se uma coleção é modificada depois que um iterador é criado para ela, o iterador se torna inválido imediatamente – as operações efetuadas com o iterador depois desse ponto lançam exceções `ConcurrentModificationException`. Por isso, diz-se que os iteradores são “fail fast” (falha rápida).*

## J.5.2 `LinkedList`

A Figura J.3 demonstra várias operações em objetos `LinkedLists`. O programa cria dois objetos `LinkedLists` de `Strings`. Os elementos de um objeto `List` são adicionados ao outro. Então, todas as `Strings` são convertidas para maiúsculas e um intervalo de elementos é excluído.

```
1 // Fig. J.3: ListTest.java
2 // Objetos List, LinkedLists e ListIterators.
3 import java.util.List;
4 import java.util.LinkedList;
5 import java.util.ListIterator;
6
7 public class ListTest
```

**Figura J.3** Objetos `List`, `LinkedLists` e `ListIterators`. (continua)



```
8 {
9 public static void main(String[] args)
10 {
11 // adiciona elementos de colors a list1
12 String[] colors =
13 { "black", "yellow", "green", "blue", "violet", "silver" };
14 List< String > list1 = new LinkedList< String >();
15
16 for (String color : colors)
17 list1.add(color);
18
19 // adiciona elementos de colors2 a list2
20 String[] colors2 =
21 { "gold", "white", "brown", "blue", "gray", "silver" };
22 List< String > list2 = new LinkedList< String >();
23
24 for (String color : colors2)
25 list2.add(color);
26
27 list1.addAll(list2); // concatena as listas
28 list2 = null; // libera recursos
29 printList(list1); // imprime elementos de list1
30
31 convertToUppercaseStrings(list1); // converte para string maiúscula
32 printList(list1); // imprime elementos de list1
33
34 System.out.print("\nDeleting elements 4 to 6...");
35 removeItems(list1, 4, 7); // remove os itens de 4 a 6 da lista
36 printList(list1); // imprime elementos de list1
37 printReversedList(list1); // imprime a lista na ordem inversa
38 } // fim de main
39
40 // gera o conteúdo da lista na saída
41 private static void printList(List< String > list)
42 {
43 System.out.println("\nlist: ");
44
45 for (String color : list)
46 System.out.printf("%s ", color);
47
48 System.out.println();
49 } // fim do método printList
50
51 // localiza objetos String e converte para maiúsculas
52 private static void convertToUppercaseStrings(List< String > list)
53 {
54 ListIterator< String > iterator = list.listIterator();
55
56 while (iterator.hasNext())
57 {
58 String color = iterator.next(); // obtém item
59 iterator.set(color.toUpperCase()); // converte para maiúscula
60 } // fim do while
61 } // fim do método convertToUppercaseStrings
62
63 // obtém sublista e usa método clear para excluir itens da sublista
64 private static void removeItems(List< String > list,
65 int start, int end)
66 {
67 list.subList(start, end).clear(); // remove itens
68 } // fim do método removeItems
69 }
```

---

**Figura J.3** Objetos List, LinkedLists e ListIterators. (continua)

```

70 // imprime a lista invertida
71 private static void printReversedList(List< String > list)
72 {
73 ListIterator< String > iterator = list.listIterator(list.size());
74
75 System.out.println("\nReversed List:");
76
77 // imprime a lista na ordem inversa
78 while (iterator.hasPrevious())
79 System.out.printf("%s ", iterator.previous());
80 } // fim do método printReversedList
81 } // fim da classe ListTest

```

```

list:
black yellow green blue violet silver gold white brown blue gray silver
list:
BLACK YELLOW GREEN BLUE VIOLET SILVER GOLD WHITE BROWN BLUE GRAY SILVER
Deleting elements 4 to 6...
list:
BLACK YELLOW GREEN BLUE WHITE BROWN BLUE GRAY SILVER
Reversed List:
SILVER GRAY BLUE BROWN WHITE BLUE GREEN YELLOW BLACK

```

**Figura J.3** Objetos List, LinkedLists e ListIterators.

As linhas 14 e 22 criam os objetos `LinkedLists` `list1` e `list2` de tipo `String`. `LinkedList` é uma classe genérica que tem um parâmetro de tipo para o qual especificamos o argumento de tipo `String` neste exemplo. As linhas 16 e 17 e também 24 e 25 chamam o método `add` de `List` para anexar elementos dos arrays `colors` e `colors2` ao final de `list1` e `list2`, respectivamente.

A linha 27 chama o **método** `addAll` de `List` para anexar todos os elementos de `list2` ao final de `list1`. A linha 28 configura `list2` como `null` para que o objeto `LinkedList` ao qual `list2` se referia possa ser pego pela coleta de lixo. A linha 29 chama o método `printList` (linhas 41 a 49) para gerar o conteúdo de `list1` na saída. A linha 31 chama o método `convertToUppercaseStrings` (linhas 52 a 61) para converter cada elemento `String` para maiúscula e, então, a linha 32 chama `printList` novamente para exibir as `Strings` modificadas. A linha 35 chama o método `removeItems` (linhas 64 a 68) para remover os elementos do índice 4 até, mas não incluindo, o índice 7 da lista. A linha 37 chama o método `printReversedList` (linhas 71 a 80) para imprimir a lista em ordem inversa.

### **Método `convertToUppercaseStrings`**

O método `convertToUppercaseStrings` (linhas 52 a 61) altera os elementos `String` minúsculos de seu argumento `List` para `Strings` maiúsculas. A linha 54 chama o **método** `listIterator` de `List` para obter o **iterador bidirecional** de `List` (isto é, um que pode percorrer o objeto `List` para trás ou para frente). `ListIterator` também é uma classe genérica. Neste exemplo, `ListIterator` faz referência a objetos `String`, pois o método `listIterator` é chamado em um objeto `List` de `Strings`. A linha 56 chama o método `hasNext` para determinar se o objeto `List` contém outro elemento. A linha 58 obtém a próxima `String` do objeto `List`. A linha 59 chama o **método** `toUpperCase` de `String` para obter uma versão em maiúsculas da `String` e chama o **método** `set` de `ListIterator` para substituir a `String` atual à qual o iterador se refere pela `String` retornada pelo método `toUpperCase`. Assim como o método `toUpperCase`, o **método** `toLowerCase` de `String` retorna uma versão em minúsculas da `String`.

### **Método `removeItems`**

O método `removeItems` (linhas 64 a 68) remove um intervalo de itens da lista. A linha 67 chama o **método `subList`** de `List` para obter uma parte do objeto `List` (chamado de **sublista**). Esse é o assim chamado **método de visualização de intervalo**, o qual permite ao programa ver uma parte da lista. A sublista é simplesmente uma visualização do objeto `List` na qual `subList` é chamado. O método `subList` recebe como argumentos o índice inicial e o final da sublista. O índice final não faz parte do intervalo da sublista. Neste exemplo, a linha 35 passa, para `subList`, 4 para o índice inicial e 7 para o final. A sublista retornada é o conjunto de elementos com os índices 4 a 6. Em seguida, o programa chama o **método `clear`** de `List` na sublista para remover os elementos da sublista do objeto `List`. Quaisquer alterações feitas em uma sublista também são feitas no objeto `List` original.

### **Método `printReversedList`**

O método `printReversedList` (linhas 71 a 80) imprime a lista de trás para a frente. A linha 73 chama o método `listIterator` de `List` com a posição inicial como argumento (em nosso caso, o último elemento da lista) para obter um iterador bidirecional para a lista. O **método `size`** de `List` retorna o número de itens presentes no objeto `List`. A condição `while` (linha 78) chama o **método `hasPrevious`** de `ListIterator` para determinar se existem mais elementos, enquanto percorre a lista de trás para a frente. A linha 79 chama o **método `previous`** de `ListIterator` para obter o elemento anterior da lista e enviá-lo para o fluxo de saída padrão.

## **J.5.3 Modos de exibição em coleções e o método `asList` de `Arrays`**

Um recurso importante do framework de coleções é a capacidade de manipular os elementos de um tipo de coleção (como um conjunto) por meio de outro tipo de coleção (como uma lista), independentemente da implementação interna da coleção. O conjunto de métodos `public` por meio dos quais as coleções são manipuladas é denominado **modo de exibição ou visualização**.

A classe `Arrays` fornece o método estático `asList` para ver um array (às vezes denominado **array de apoio**) como uma coleção `List`. Um modo de exibição de `List` permite a manipulação do array como se ele fosse uma lista. Isso é útil para adicionar os elementos de um array a uma coleção e para ordenar os elementos do array. O próximo exemplo demonstra como criar um objeto `LinkedList` com um modo de exibição de `List` de um array, pois não podemos passar o array para um construtor de `LinkedList`. Quaisquer modificações feitas por meio do modo de exibição de `List` alteram o array, e quaisquer modificações feitas no array alteram o modo de exibição de `List`. A única operação permitida no modo de exibição retornado por `asList` é `set`, a qual altera o valor do modo de exibição e o array de apoio. Quaisquer outras tentativas de alterar o modo de exibição (como adicionar ou remover elementos) resulta em uma exceção `UnsupportedOperationException`.

### **Visualização de arrays como objetos `List` e conversão de objetos `List` em arrays**

A Figura J.4 usa o método `asList` de `Arrays` para visualizar um array como um objeto `List` e usa o **método `toArray`** de `List` para obter um array a partir de uma coleção `LinkedList`. O programa chama o método `asList` para criar um modo de exibição de `List` de um array, o qual é usado para inicializar um objeto `LinkedList`; então, adiciona uma série de strings a `LinkedList` e chama o método `toArray` para obter um array contendo referências para as Strings.

```

1 // Fig. J.4: UsingToArray.java
2 // Visualização de arrays como objetos List e conversão de objetos List em arrays.
3 import java.util.LinkedList;
4 import java.util.Arrays;
5
6 public class UsingToArray
7 {
8 // cria um objeto LinkedList, adiciona elementos e converte em array
9 public static void main(String[] args)
10 {
11 String[] colors = { "black", "blue", "yellow" };
12
13 LinkedList< String > links =
14 new LinkedList< String >(Arrays.asList(colors));
15
16 links.addLast("red"); // adiciona como último item
17 links.add("pink"); // adiciona no final
18 links.add(3, "green"); // adiciona no 3º índice
19 links.addFirst("cyan"); // adiciona como primeiro item
20
21 // obtém os elementos de LinkedList como um array
22 colors = links.toArray(new String[links.size()]);
23
24 System.out.println("colors: ");
25
26 for (String color : colors)
27 System.out.println(color);
28 } // fim de main
29 } // fim da classe UsingToArray

```

```

colors:
cyan
black
blue
yellow
green
red
pink

```

**Figura J.4** Visualização de arrays como objetos List e conversão de objetos List em arrays.

As linhas 13 e 14 constroem um objeto `LinkedList` de Strings contendo os elementos do array `colors`. A linha 14 usa o método `asList` de `Arrays` para retornar um modo de exibição de `List` do array e, então, usa isso para inicializar o objeto `LinkedList` com seu construtor, que recebe um objeto `Collection` como argumento (um objeto `List` é um objeto `Collection`). A linha 16 chama o **método** `addLast` de `LinkedList` para adicionar "red" ao final de `links`. As linhas 17 e 18 chamam o **método** `add` de `LinkedList` para adicionar "pink" como último elemento e "green" como o elemento no índice 3 (isto é, o quarto elemento). O método `addLast` (linha 16) funciona exatamente como o método `add` (linha 17). A linha 19 chama o **método** `addFirst` de `LinkedList` para adicionar "cyan" como o novo primeiro item no objeto `LinkedList`. As operações `add` são permitidas porque atuam sobre o objeto `LinkedList`, não no modo de exibição retornado por `asList`.

A linha 22 chama o método `toArray` da interface `List` para obter um array de Strings a partir de `links`. O array é uma cópia dos elementos da lista – modificar o conteúdo do array *não* modifica a lista. O array passado para o método `toArray` é do mesmo tipo que você queria que ele retornasse. Se o número de elementos nesse array é maior

ou igual ao número de elementos no objeto `LinkedList`, `toArray` copia os elementos da lista no argumento de seu array e retorna esse array. Se o objeto `LinkedList` tem mais elementos que o número de elementos no array passado para `toArray`, `toArray` aloca um novo array do mesmo tipo que recebe como argumento, copia os elementos da lista no novo array e retorna o novo array.

## J.6 Métodos de Collections

A classe `Collections` fornece vários algoritmos de alto desempenho (Fig. J.5) para manipular elementos de coleção. Os algoritmos são implementados como métodos estáticos. Os métodos `sort`, `binarySearch`, `reverse`, `shuffle`, `fill` e `copy` operam em objetos `List`. Os métodos `min`, `max` e `addAll` operam em objetos `Collection`.

| Método                    | Descrição                                                                             |
|---------------------------|---------------------------------------------------------------------------------------|
| <code>sort</code>         | Ordena os elementos de uma lista.                                                     |
| <code>binarySearch</code> | Localiza um objeto em uma lista.                                                      |
| <code>reverse</code>      | Inverte os elementos de uma lista.                                                    |
| <code>shuffle</code>      | Ordena os elementos de uma lista aleatoriamente.                                      |
| <code>fill</code>         | Configura cada elemento da lista de modo a fazer referência a um objeto especificado. |
| <code>copy</code>         | Copia as referências de uma lista para outra.                                         |
| <code>min</code>          | Retorna o menor elemento de uma coleção.                                              |
| <code>max</code>          | Retorna o maior elemento de uma coleção.                                              |
| <code>addAll</code>       | Anexa todos os elementos de um array a uma coleção.                                   |

**Figura J.5** Alguns métodos da classe `Collections`.

### J.6.1 Método `sort`

O **método** `sort` ordena os elementos de um objeto `List`, o qual deve implementar a **interface** `Comparable`. A ordem é determinada pela ordem natural do tipo dos elementos, conforme implementado pelo método `compareTo`. O método `compareTo` é declarado na interface `Comparable` e às vezes é chamado de **método de comparação natural**. A chamada de `sort` pode especificar um objeto `Comparator` como segundo argumento, o qual determina uma ordem alternativa para os elementos.

#### **Classificando em ordem crescente ou decrescente**

Se a lista é um objeto `List` de objetos `Comparable` (como `Strings`), você pode usar o método `sort` de `Collections` para classificar os elementos em ordem crescente, como segue:

```
Collections.sort(list); // classifica list em ordem crescente
```

Para classificar `List` em ordem decrescente, use isto:

```
// classifica list em ordem decrescente
Collections.sort(list, Collections.reverseOrder());
```

O **método** estático `reverseOrder` de `Collections` retorna um objeto `Comparator` que classifica os elementos da coleção em ordem inversa.

## Ordenando com Comparator

Para objetos que não são Comparable, você pode criar objetos Comparator personalizados. A Figura J.6 cria uma classe Comparator personalizada, chamada TimeComparator, que implementa a interface Comparator para comparar dois objetos Time2. A classe Time2, declarada na Fig. F.5, representa o tempo como horas, minutos e segundos.

```

1 // Fig. J.8: TimeComparator.java
2 // Classe Comparator personalizada que compara dois objetos Time2.
3 import java.util.Comparator;
4
5 public class TimeComparator implements Comparator< Time2 >
6 {
7 public int compare(Time2 time1, Time2 time2)
8 {
9 int hourCompare = time1.getHour() - time2.getHour(); // compara a hora
10
11 // testa a hora primeiro
12 if (hourCompare != 0)
13 return hourCompare;
14
15 int minuteCompare =
16 time1.getMinute() - time2.getMinute(); // compara o minuto
17
18 // então, testa o minuto
19 if (minuteCompare != 0)
20 return minuteCompare;
21
22 int secondCompare =
23 time1.getSecond() - time2.getSecond(); // compara o segundo
24
25 return secondCompare; // retorna o resultado da comparação dos segundos
26 } // fim do método compare
27 } // fim da classe TimeComparator

```

**Figura J.6** Classe Comparator personalizada que compara dois objetos Time2.

A classe TimeComparator implementa a interface Comparator, um tipo genérico que recebe um único argumento de tipo (neste caso, Time2). Uma classe que implementa Comparator deve declarar um método compare que receba dois argumentos e retorne um valor inteiro negativo se o primeiro argumento for menor que o segundo, 0 se os argumentos forem iguais ou um valor inteiro positivo se o primeiro argumento for maior que o segundo. O método compare (linhas 7 a 26) faz comparações entre objetos Time2. A linha 9 compara as duas horas dos objetos Time2. Se forem diferentes (linha 12), então retornamos esse valor. Se esse valor for positivo, então a primeira hora é maior que a segunda e o primeiro tempo é maior que o segundo. Se esse valor for negativo, então a primeira hora é menor que a segunda e o primeiro tempo é menor que o segundo. Se esse valor for zero, as horas são as mesmas e devemos testar os minutos (e talvez os segundos) para determinar qual tempo é maior.

A Figura J.7 ordena uma lista usando a classe TimeComparator personalizada de Comparator. A linha 11 cria uma ArrayList de objetos Time2. Lembre-se de que tanto ArrayList como List são tipos genéricos e aceitam um argumento de tipo que especifica o tipo de elemento da coleção. As linhas 13 a 17 criam cinco objetos Time2 e os adicionam a essa lista. A linha 23 chama o método sort, passando-o como um objeto de nossa classe TimeComparator (Fig. J.6).

```

1 // Fig. J.7: Sort.java
2 // Método sort de Collections com um objeto Comparator personalizado.
3 import java.util.List;
4 import java.util.ArrayList;
5 import java.util.Collections;
6
7 public class Sort3
8 {
9 public static void main(String[] args)
10 {
11 List< Time2 > list = new ArrayList< Time2 >(); // cria List
12
13 list.add(new Time2(6, 24, 34));
14 list.add(new Time2(18, 14, 58));
15 list.add(new Time2(6, 05, 34));
16 list.add(new Time2(12, 14, 58));
17 list.add(new Time2(6, 24, 22));
18
19 // gera os elementos de List
20 System.out.printf("Unsorted array elements:\n%s\n", list);
21
22 // classifica em ordem usando um comparador
23 Collections.sort(list, new TimeComparator());
24
25 // gera os elementos de List
26 System.out.printf("Sorted list elements:\n%s\n", list);
27 } // fim de main
28 } // fim da classe Sort3

```

```

Unsorted array elements:
[6:24:34 AM, 6:14:58 PM, 6:05:34 AM, 12:14:58 PM, 6:24:22 AM]
Sorted list elements:
[6:05:34 AM, 6:24:22 AM, 6:24:34 AM, 12:14:58 PM, 6:14:58 PM]

```

**Figura J.7** Método sort de Collections com um objeto Comparator personalizado.

## J.6.2 Método shuffle

O método **shuffle** ordena os elementos de um objeto **List** aleatoriamente. O Apêndice E apresentou uma simulação de embaralhamento e distribuição de cartas que misturava um baralho de cartas com um loop. Se você tem um array com 52 objetos **Card**, pode embaralhá-los com o método **shuffle**, como segue:

```

List< Card > list = Arrays.asList(deck); // obtém List
Collections.shuffle(list); // embaralha as cartas

```

A segunda linha acima embaralha o array chamando o método estático **shuffle** da classe **Collections**. O método **shuffle** exige um argumento **List**; portanto, devemos obter um modo de exibição de **List** do array antes de podermos embaralhá-lo. O método estático **asList** da classe **Arrays** obtém um modo de exibição de **List** do array **deck**.

## J.7 Interface Queue

Uma fila é uma coleção que representa uma fila de espera – normalmente, as inserções são feitas no final de uma fila e as exclusões a partir do início. A interface **Queue** estende a interface **Collection** e fornece operações adicionais para inserir, remover e inspecionar

os elementos de uma fila. Você pode ver os detalhes da interface `Queue` e a lista de classes que a implementam em

[docs.oracle.com/javase/6/docs/api/index.html?java/util/Queue.html](https://docs.oracle.com/javase/6/docs/api/index.html?java/util/Queue.html)

## J.8 Conjuntos

**Set** (conjunto) é um objeto `Collection` não ordenado de elementos exclusivos (isto é, sem elementos duplicados). O framework de coleções contém várias implementações de `Set`, incluindo `HashSet` e `TreeSet`. `HashSet` armazena seus elementos em uma tabela hash, e `TreeSet` armazena seus elementos em uma árvore. As tabelas hash são apresentadas na seção J.9.

A Figura J.8 usa um objeto `HashSet` para remover strings duplicadas de um objeto `List`. Lembre-se de que tanto `List` como `Collection` são tipos genéricos, de modo que a linha 16 cria uma lista contendo objetos `String`, e a linha 20 passa uma coleção de `Strings` para o método `printNonDuplicates`.

```

1 // Fig. J.8: SetTest.java
2 // HashSet usado para remover valores duplicados de um array de strings.
3 import java.util.List;
4 import java.util.Arrays;
5 import java.util.HashSet;
6 import java.util.Set;
7 import java.util.Collection;
8
9 public class SetTest
10 {
11 public static void main(String[] args)
12 {
13 // cria e exibe um objeto List< String >
14 String[] colors = { "red", "white", "blue", "green", "gray",
15 "orange", "tan", "white", "cyan", "peach", "gray", "orange" };
16 List< String > list = Arrays.asList(colors);
17 System.out.printf("List: %s\n", list);
18
19 // elimina as duplicatas e então imprime os valores únicos
20 printNonDuplicates(list);
21 } // fim de main
22
23 // cria um conjunto a partir de uma coleção para eliminar duplicatas
24 private static void printNonDuplicates(Collection< String > values)
25 {
26 // cria um HashSet
27 Set< String > set = new HashSet< String >(values);
28
29 System.out.print("\nNonduplicates are: ");
30
31 for (String value : set)
32 System.out.printf("%s ", value);
33
34 System.out.println();
35 } // fim do método printNonDuplicates
36 } // fim da classe SetTest

```

List: [red, white, blue, green, gray, orange, tan, white, cyan, peach, gray, orange]

Nonduplicates are: orange green white peach gray cyan red blue tan

**Figura J.8** `HashSet` usado para remover valores duplicados de um array de strings.



O método `printNonDuplicates` (linhas 24 a 35) recebe um argumento `Collection`. A linha 27 constrói um `HashSet<String>` a partir do argumento `Collection<String>`. Por definição, os objetos `Set` não contêm duplicatas; portanto, quando o `HashSet` é construído, ele remove quaisquer duplicatas no objeto `Collection`. As linhas 31 e 32 geram os elementos do objeto `Set`.

### Conjuntos ordenados

O framework de coleções inclui também a **interface `SortedSet`** (que estende `Set`) para conjuntos que mantêm seus elementos classificados – ou na ordem natural dos elementos (por exemplo, números em ordem crescente) ou em uma ordem especificada por um objeto `Comparator`. A classe `TreeSet` implementa `SortedSet`. Os itens colocados em um objeto `TreeSet` são ordenados à medida que são adicionados.

## J.9 Mapas

**Maps** (Mapas) associam chaves a valores. As chaves de um mapa devem ser exclusivas, mas os valores associados não precisam ser. Se um mapa contém chaves e valores exclusivos, diz-se que ele implementa um **mapeamento de um para um**. Se apenas as chaves são exclusivas, diz-se que o mapa implementa um **mapeamento de muitos para um** – muitas chaves podem ser mapeadas para um único valor.

A diferença entre os mapas (Maps) e os conjuntos (Sets) é que os mapas contêm chaves e valores, enquanto os conjuntos contêm apenas valores. Três das várias classes que implementam a interface `Map` são **`Hashtable`**, **`HashMap`** e **`TreeMap`**, e os mapas são extensivamente usados no Android. `HashTables` e `HashMaps` armazenam elementos em tabelas hash, e `TreeMaps` armazenam elementos em árvores – os detalhes das estruturas de dados subjacentes estão fora dos objetivos deste livro. A **interface `SortedMap`** estende `Map` e mantém suas chaves ordenadas – ou na ordem natural dos elementos ou em uma ordem especificada por um objeto `Comparator`. A classe `TreeMap` implementa `SortedMap`. A Figura J.9 usa um `HashMap` para contar o número de ocorrências de cada palavra em uma string.

```

1 // Fig. J.9: WordTypeCount.java
2 // O programa conta o número de ocorrências de cada palavra em uma String.
3 import java.util.Map;
4 import java.util.HashMap;
5 import java.util.Set;
6 import java.util.TreeSet;
7 import java.util.Scanner;
8
9 public class WordTypeCount
10 {
11 public static void main(String[] args)
12 {
13 // cria HashMap para armazenar chaves String e valores Integer
14 Map< String, Integer > myMap = new HashMap< String, Integer >();
15
16 createMap(myMap); // cria mapa baseado na entrada do usuário
17 displayMap(myMap); // exibe o conteúdo do mapa
18 } // fim de main
19
20 // cria mapa a partir da entrada do usuário
21 private static void createMap(Map< String, Integer > map)
22 {

```

**Figura J.9** O programa conta o número de ocorrências de cada palavra em uma String.

```

23 Scanner scanner = new Scanner(System.in); // cria scanner
24 System.out.println("Enter a string:"); // solicita entrada do usuário
25 String input = scanner.nextLine();
26
27 // transforma a entrada em tokens
28 String[] tokens = input.split(" ");
29
30 // processamento do texto de entrada
31 for (String token : tokens)
32 {
33 String word = token.toLowerCase(); // obtém palavra em minúsculas
34
35 // se o mapa contém a palavra
36 if (map.containsKey(word)) // se a palavra está no mapa
37 {
38 int count = map.get(word); // obtém a contagem atual
39 map.put(word, count + 1); // incrementa a contagem
40 } // fim do if
41 else
42 map.put(word, 1); // adiciona ao mapa a nova palavra com contagem 1
43 } // fim do for
44 } // fim do método createMap
45
46 // exibe o conteúdo do mapa
47 private static void displayMap(Map< String, Integer > map)
48 {
49 Set< String > keys = map.keySet(); // obtém as chaves
50
51 // ordena as chaves
52 TreeSet< String > sortedKeys = new TreeSet< String >(keys);
53
54 System.out.println("\nMap contains:\nKey\t\tValue");
55
56 // gera a saída para cada chave do mapa
57 for (String key : sortedKeys)
58 System.out.printf("%-10s%10s\n", key, map.get(key));
59
60 System.out.printf(
61 "\nsize: %d\nisEmpty: %b\n", map.size(), map.isEmpty());
62 } // fim do método displayMap
63 } // fim da classe WordTypeCount

```

```

Enter a string:
this is a sample sentence with several words this is another sample
sentence with several different words

Map contains:
Key Value
a 1
another 1
different 1
is 2
sample 2
sentence 2
several 2
this 2
with 2
words 2

size: 10
isEmpty: false

```

**Figura J.9** O programa conta o número de ocorrências de cada palavra em uma String.

A linha 14 cria um `HashMap` vazio com capacidade inicial padrão (16 elementos) e um fator de carga padrão (0.75) – esses padrões são incorporados à implementação de `HashMap`. Quando o número de vagas ocupadas no `HashMap` se torna maior que a capacidade vezes o fator de carga, a capacidade é duplicada automaticamente. `HashMap` é uma classe genérica que recebe dois argumentos de tipo – o tipo da chave (isto é, `String`) e o tipo do valor (isto é, `Integer`). Lembre-se de que os argumentos de tipo passados para uma classe genérica devem ser tipos de referência, por isso o segundo argumento de tipo é `Integer` e não `int`.

A linha 16 chama o método `createMap` (linhas 21 a 44), o qual usa um mapa para armazenar o número de ocorrências de cada palavra na frase. A linha 25 obtém a entrada do usuário e a linha 28 a transforma em tokens. O loop das linhas 31 a 43 converte o próximo token em letras maiúsculas (linha 33) e, então, chama o **método** `containsKey` de `Map` (linha 36) para determinar se a palavra está no mapa (e, assim, ocorreu anteriormente na string). Se o objeto `Map` não contém um mapeamento para a palavra, a linha 42 usa o **método** `put` de `Map` para criar uma nova entrada no mapa, com a palavra como chave e um objeto `Integer` contendo 1 como valor. O autoboxing ocorre quando o programa passa o valor inteiro 1 para o método `put`, pois o mapa armazena o número de ocorrências da palavra como um valor `Integer`. Se a palavra existe no mapa, a linha 38 usa o **método** `get` de `Map` para obter o valor associado da chave (a contagem) no mapa. A linha 39 incrementa esse valor e usa `put` para substituir o valor associado à chave no mapa. O método `put` retorna o valor associado anterior da chave ou `null`, se a chave não estava no mapa.

O método `displayMap` (linhas 47 a 62) exibe todas as entradas do mapa. Ele usa o **método** `keySet` de `HashMap` (linha 49) para obter um conjunto das chaves. As chaves têm tipo `String` no mapa, de modo que o método `keySet` retorna um objeto `Set` de tipo genérico com o parâmetro de tipo especificado como `String`. A linha 52 cria um objeto `TreeSet` das chaves, no qual elas são ordenadas. O loop nas linhas 57 e 58 acessa cada chave e seu valor no mapa. A linha 58 exibe cada chave e seu valor usando o especificador de formato `%-10s` para alinhar cada chave à esquerda, e o especificador de formato `%10s` para alinhar cada valor à direita. As chaves são exibidas em ordem crescente. A linha 61 chama o **método** `size` de `Map` para obter o número de pares chave/valor no objeto `Map`. A linha 61 também chama o **método** `isEmpty` de `Map`, o qual retorna um valor `boolean` indicando se o objeto `Map` está vazio.

## J.10 Introdução aos arquivos e fluxos

Os dados armazenados em variáveis e arrays são temporários – eles são perdidos quando uma variável local sai do escopo ou quando o programa termina. Para retenção de dados em longo prazo, mesmo depois que os programas que os criam terminam, os computadores usam **arquivos**. Você usa arquivos diariamente para tarefas como escrever um documento ou criar uma planilha. Os dados mantidos em arquivos são **persistentes** – eles existem além da duração da execução do programa.

### *Arquivos como fluxos de bytes*

A linguagem Java enxerga cada arquivo como um **fluxo de bytes** sequencial (Fig. J.10). Todo sistema operacional fornece um mecanismo para determinar o fim de um arquivo, como um **marcador de fim de arquivo** ou uma contagem do total de bytes no arquivo que é gravado em uma estrutura de dados administrativa mantida pelo sistema. Um

programa Java processando um fluxo de bytes simplesmente recebe uma indicação do sistema operacional ao atingir o final do fluxo – o programa *não* precisa saber como a plataforma subjacente representa arquivos ou fluxos. Em alguns casos, a indicação de fim de arquivo ocorre como uma exceção. Em outros, a indicação é um valor de retorno de um método chamado em um objeto de processamento de fluxo.



**Figura J.10** Visão da linguagem Java de um arquivo de  $n$  bytes.

### Fluxos baseados em bytes e baseados em caracteres

Os fluxos podem ser usados para gerar entrada e saída de dados como bytes ou como caracteres. Os **fluxos baseados em bytes** geram entrada e saída de dados em seu formato binário. Os **fluxos baseados em caracteres** geram entrada e saída de dados como uma sequência de caracteres. Se o valor 5 fosse armazenado utilizando-se um fluxo baseado em bytes, seria armazenado no formato binário do valor numérico 5, ou 101. Se o valor 5 fosse armazenado utilizando-se um fluxo baseado em caracteres, seria armazenado no formato binário do caractere 5, ou 00000000 00110101 (essa é a representação binária do valor numérico 53, a qual indica o caractere 5 em Unicode). A diferença entre as duas formas é que o valor numérico pode ser usado como um valor inteiro em cálculos, enquanto o caractere 5 é simplesmente um caractere que pode ser usado em uma string de texto, como em "Sarah Miller tem 15 anos". Os arquivos criados usando-se fluxos baseados em bytes são referidos como **arquivos binários**, enquanto os arquivos criados usando-se fluxos baseados em caracteres são referidos como **arquivos de texto**. Os arquivos de texto podem ser lidos por editores de texto, enquanto os arquivos binários são lidos por programas que entendem o conteúdo específico do arquivo e sua ordenação.

### Abrindo um arquivo

Um programa Java **abre** um arquivo criando um objeto e associando a ele um fluxo de bytes ou de caracteres. O construtor do objeto interage com o sistema operacional para abrir o arquivo.

### O pacote `java.io`

Os programas Java realizam processamento de arquivos usando classes do pacote `java.io`. Esse pacote inclui definições para classes de fluxo, como `FileInputStream` (para entrada baseada em bytes a partir de um arquivo), `FileOutputStream` (para saída baseada em bytes em um arquivo), `FileReader` (para entrada baseada em caracteres a partir de um arquivo) e `FileWriter` (para saída baseada em caracteres em um arquivo), as quais herdam das classes `InputStream`, `OutputStream`, `Reader` e `Writer`, respectivamente. Assim, os métodos dessas classes de fluxo podem ser aplicados a fluxos de arquivo.

A linguagem Java contém classes que permitem realizar entrada e saída de objetos ou variáveis de tipos de dados primitivos. Nos bastidores, os dados ainda serão armazenados como bytes ou caracteres, permitindo que você leia ou escreva dados na forma de valores `int`, `String` ou outros tipos, sem precisar se preocupar com os detalhes da conversão de tais valores para o formato de bytes. Para fazer essa entrada e saída, os objetos das

classes `ObjectInputStream` e `ObjectOutputStream` podem ser usados junto com as classes de fluxo baseadas em bytes `FileInputStream` e `FileOutputStream` (essas classes vão ser discutidas com mais detalhes em breve). A hierarquia completa de tipos do pacote `java.io` pode ser vista na documentação online, em

[docs.oracle.com/javase/6/docs/api/java/io/package-tree.html](http://docs.oracle.com/javase/6/docs/api/java/io/package-tree.html)

Entrada e saída baseadas em caracteres também podem ser feitas com as classes `Scanner` e `Formatter`. A classe `Scanner` é muito usada para entrada de dados a partir do teclado – ela também pode ler dados de um arquivo. A classe `Formatter` permite gerar dados formatados para qualquer fluxo baseado em texto, de maneira semelhante ao método `System.out.printf`.

## J.11 Classe File

A classe serve para recuperar informações sobre arquivos ou diretórios do disco. Os objetos `File` são frequentemente usados com objetos de outras classes `java.io` para especificar arquivos ou diretórios a manipular.

### *Criando objetos File*

A classe `File` fornece vários construtores. O que tem um argumento `String` especifica o nome de um arquivo ou diretório a ser associado ao objeto `File`. O nome pode conter **informações de caminho** e o nome de um arquivo ou diretório. O caminho de um arquivo ou diretório especifica sua localização no disco. O caminho inclui alguns ou todos os diretórios que levam ao arquivo ou diretório. Um **caminho absoluto** contém todos os diretórios, começando no **diretório-raiz**, que levam a um arquivo ou diretório específico. Todo arquivo ou diretório de uma unidade de disco em particular tem o mesmo diretório-raiz em seu caminho. Um **caminho relativo** normalmente começa no diretório em que o aplicativo começou a ser executado e, portanto, é “relativo” ao diretório atual. O construtor de dois argumentos `String` especifica um caminho absoluto ou relativo como primeiro argumento e, como segundo argumento, o arquivo ou diretório a associar ao objeto `File`. O construtor com argumentos `File` e `String` usa um objeto `File` já existente descrevendo o diretório pai do arquivo ou diretório especificado pelo argumento `String`. O quarto construtor utiliza um objeto `URI` para localizar o arquivo. Um **URI (Uniform Resource Identifier)** é uma forma mais geral dos **URLs (Uniform Resource Locators)** utilizados para localizar sites.

Por exemplo, `http://www.deitel.com/` é o URL do site da Deitel & Associates. Os URIs para localizar arquivos variam entre os sistemas operacionais. Nas plataformas Windows, o URI

`file://C:/data.txt`

identifica o arquivo `data.txt` armazenado no diretório-raiz da unidade C:. Nas plataformas UNIX/Linux, o URI

`file:/home/student/data.txt`

identifica o arquivo `data.txt` armazenado no diretório de base do usuário `student`.

A Figura J.11 lista alguns métodos comuns de `File`. A lista completa pode ser vista em [docs.oracle.com/javase/6/docs/api/java/io/File.html](http://docs.oracle.com/javase/6/docs/api/java/io/File.html).

| Método                                | Descrição                                                                                                                                                                                                               |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>boolean canRead()</code>        | Retorna <code>true</code> se um arquivo pode ser lido pelo aplicativo atual; caso contrário, retorna <code>false</code> .                                                                                               |
| <code>boolean canWrite()</code>       | Retorna <code>true</code> se um arquivo pode ser escrito pelo aplicativo atual; caso contrário, retorna <code>false</code> .                                                                                            |
| <code>boolean exists()</code>         | Retorna <code>true</code> se o arquivo ou diretório representado pelo objeto <code>File</code> existe; caso contrário, retorna <code>false</code> .                                                                     |
| <code>boolean isFile()</code>         | Retorna <code>true</code> se o nome especificado como argumento para o construtor de <code>File</code> é um arquivo; caso contrário, retorna <code>false</code> .                                                       |
| <code>boolean isDirectory()</code>    | Retorna <code>true</code> se o nome especificado como argumento para o construtor de <code>File</code> é um diretório; caso contrário, retorna <code>false</code> .                                                     |
| <code>boolean isAbsolute()</code>     | Retorna <code>true</code> se os argumentos especificados para o construtor de <code>File</code> indicam um caminho absoluto para um arquivo ou diretório; caso contrário, retorna <code>false</code> .                  |
| <code>String getAbsolutePath()</code> | Retorna uma <code>String</code> com o caminho absoluto do arquivo ou diretório.                                                                                                                                         |
| <code>String getName()</code>         | Retorna uma <code>String</code> com o nome do arquivo ou diretório.                                                                                                                                                     |
| <code>String getPath()</code>         | Retorna uma <code>String</code> com o caminho do arquivo ou diretório.                                                                                                                                                  |
| <code>String getParent()</code>       | Retorna uma <code>String</code> com o diretório pai do arquivo ou diretório (isto é, o diretório no qual o arquivo ou diretório está localizado).                                                                       |
| <code>long length()</code>            | Retorna o comprimento do arquivo em bytes. Se o objeto <code>File</code> representa um diretório, é retornado um valor não especificado.                                                                                |
| <code>long lastModified()</code>      | Retorna uma representação dependente de plataforma da hora em que o arquivo ou diretório foi modificado pela última vez. O valor retornado é útil apenas para comparação com outros valores retornados por esse método. |
| <code>String[] list()</code>          | Retorna um array de <code>Strings</code> representando o conteúdo de um diretório. Retorna <code>null</code> se o objeto <code>File</code> não representa um diretório.                                                 |

Figura J.11 Métodos de `File`.

## J.12 Introdução à serialização de objetos

A linguagem Java fornece a **serialização de objetos** para escrever objetos inteiros em um fluxo e ler objetos inteiros de um fluxo. Um assim chamado **objeto serializado** é representado como uma sequência de bytes que incluem os dados do objeto e informações sobre o tipo do objeto e os tipos dos dados nele armazenados. Depois que um objeto serializado é escrito em um arquivo, ele pode ser **desserializado** – isto é, as informações de tipo e os bytes que representam o objeto e seus dados podem ser usados para recriar o objeto na memória.

### **Classes `ObjectInputStream` e `ObjectOutputStream`**

As classes `ObjectInputStream` e `ObjectOutputStream`, que respectivamente implementam as interfaces **`ObjectInput`** e **`ObjectOutput`**, permitem que objetos por inteiro sejam lidos de ou escritos em um fluxo (possivelmente um arquivo). Para utilizar serialização com arquivos, inicializamos objetos `ObjectInputStream` e `ObjectOutputStream` com objetos fluxo que leem de e escrevem em arquivos – objetos das classes `FileInputStream` e `FileOutputStream`, respectivamente. A inicialização de objetos fluxo com outros objetos fluxo dessa maneira às vezes é chamada de **empacotamento** – o novo objeto fluxo que está sendo criado empacota o objeto fluxo especificado como argumento do construtor. Para empacotar um objeto `FileInputStream` em um objeto `ObjectInputStream`, por exemplo, passamos o objeto `FileInputStream` para o construtor de `ObjectInputStream`.

### Interfaces *ObjectOutput* e *ObjectInput*

A interface *ObjectOutput* contém o método `writeObject`, o qual recebe um *Object* como argumento e escreve suas informações em um objeto *OutputStream*. Uma classe que implemente a interface *ObjectOutput* (como *ObjectOutputStream*) declara esse método e garante que o objeto que está sendo gerado implementa a interface *Serializable* (a ser discutida em breve). De forma correspondente, a interface *ObjectInput* contém o método `readObject`, o qual lê e retorna uma referência para um *Object* a partir de um objeto *InputStream*. Depois que um objeto é lido, sua referência pode ser convertida para o tipo efetivo do objeto.

## J.13 Introdução ao multithread

Seria ótimo se pudéssemos concentrar nossa atenção na execução de apenas uma ação por vez e executá-la bem, mas normalmente isso é difícil. O corpo humano executa uma grande variedade de operações *em paralelo* – ou, como dizemos no mundo da programação, **de forma concorrente**. Respiração, circulação sanguínea, digestão, pensamento e o caminhar, por exemplo, podem ocorrer concomitantemente (ou de forma concorrente), assim como todos os sentidos – visão, tato, olfato, paladar e audição.

Os computadores também podem executar operações de forma concorrente. É comum os computadores pessoais compilarem um programa, enviarem um arquivo para uma impressora e receberem mensagens de correio eletrônico por meio de uma rede de forma concorrente. Somente os computadores que possuem vários processadores podem verdadeiramente executar várias instruções concomitantemente. Nos computadores de processador único, os sistemas operacionais criam a ilusão de execução concorrente alternando entre as atividades rapidamente, mas nesses computadores apenas uma instrução pode ser executada por vez. Os computadores multinúcleo atuais têm vários processadores que permitem executar tarefas de forma verdadeiramente concorrente. Além disso, estão começando a surgir smartphones multinúcleo.

### Concorrência na linguagem Java

O Java torna a concorrência acessível por meio da linguagem e de APIs. Os programas Java podem ter várias **threads de execução**, onde cada thread tem sua própria pilha de chamada de métodos e seu próprio contador de programa, permitindo ser executada concomitantemente com outras threads, enquanto compartilha com elas os recursos em nível de aplicativo, como a memória. Essa capacidade é chamada de **multithread**.



#### Dica de desempenho J.1

*Um problema dos aplicativos de thread única que pode levar à lentidão de respostas é que as atividades prolongadas precisam terminar antes que outras possam começar. Em um aplicativo multithread, as threads podem ser distribuídas por vários processadores (se disponíveis), de modo que várias tarefas são executadas de forma verdadeiramente concorrente e o aplicativo pode funcionar de modo mais eficiente. O multithread também pode aumentar o desempenho em sistemas de processador único que simulam a concorrência – quando uma thread não pode prosseguir (porque, por exemplo, está esperando pelo resultado de uma operação de E/S), outra pode usar o processador.*

### Usos da programação concorrente

Vamos discutir muitas aplicações da **programação concorrente**. Por exemplo, ao fazer download de um arquivo grande (por exemplo, uma imagem, um clipe de áudio ou um videoclipe) pela Internet, talvez o usuário não queira esperar até que o clip seja baixado

para então iniciar a reprodução. Para resolver esse problema, várias threads podem ser usadas – uma para baixar o clipe e outra para reproduzi-lo. Essas atividades ocorrem de forma concorrente. Para evitar uma reprodução picada, as threads são **sincronizadas** (isto é, suas ações são coordenadas) para que o reproduzidor não comece até que haja um volume suficiente do clipe na memória para manter sua thread ocupada. A Java Virtual Machine (JVM) cria threads para executar programas e threads para realizar tarefas de limpeza, como a coleta de lixo.

### **Programação concorrente é difícil**

Escrever programas multithread pode ser complicado. Embora o cérebro humano possa executar funções de forma concorrente, as pessoas acham difícil saltar entre linhas de pensamento paralelas. Para ver por que pode ser difícil escrever e entender programas multithread, tente a seguinte experiência: abra três livros na página 1 e tente lê-los concomitantemente. Leia algumas palavras do primeiro livro, depois algumas do segundo, algumas do terceiro e então volte e leia as palavras seguintes do primeiro e assim por diante. Depois dessa experiência, você vai reconhecer muitos dos desafios do multithread – alternar entre os livros, ler brevemente, lembrar-se de seu lugar em cada livro, mover o livro que você está lendo para mais perto a fim de que possa lê-lo, não prestar atenção nos livros que não está lendo e, no meio de todo esse caos, tentar compreender o conteúdo dos livros!

### **Use as classes predefinidas das APIs de concorrência quando possível**

A programação de aplicativos concorrentes é difícil e propensa a erros. Se for preciso usar sincronização em um programa, você deve *usar as classes existentes das APIs de concorrência que gerenciam a sincronização*. Essas classes são escritas por especialistas, foram completamente testadas e depuradas, funcionam de forma eficiente e o ajudam a evitar as armadilhas e ciladas comuns.

## **J.14 Criação e execução de threads com o framework Executor**

Esta seção demonstra como executar tarefas concorrentes em um aplicativo usando objetos Executors e Runnable.

### **Criando tarefas concorrentes com a interface Runnable**

Você implementa a interface **Runnable** (do pacote `java.lang`) para especificar uma tarefa que pode ser executada de forma concorrente com outras tarefas. A interface **Runnable** declara o único método **run**, o qual contém o código que define a tarefa a ser executada por um objeto **Runnable**.

### **Executando objetos Runnable com um objeto Executor**

Para permitir que um objeto **Runnable** faça sua tarefa, você deve executá-lo. Um objeto **Executor** executa objetos **Runnable**. Ele faz isso criando e gerenciando um grupo de threads chamado **pool de threads**. Quando um objeto **Executor** começa a executar um objeto **Runnable**, o **Executor** chama o método **run** do objeto **Runnable**, o qual é executado em uma nova thread.

A interface **Executor** declara um único método, chamado **execute**, que recebe um objeto **Runnable** como argumento. O objeto **Executor** atribui cada objeto **Runnable** passado para seu método **execute** para uma das threads disponíveis no pool de threads. Se



não há threads disponíveis, o objeto `Executor` cria uma nova ou espera que uma thread fique livre e atribui essa thread ao objeto `Runnable` passado para o método `execute`.

É muito mais vantajoso usar um objeto `Executor` do que você mesmo criar threads. Os objetos `Executor` podem *reutilizar threads existentes* para eliminar a sobrecarga da criação de uma nova thread para cada tarefa e podem aumentar o desempenho, *otimizando o número de threads* para garantir que o processador permaneça ocupado, sem criar tantas threads que o aplicativo fique sem recursos.



### Observação sobre engenharia de software J.2

*Embora seja possível criar threads explicitamente, recomenda-se usar a interface `Executor` para gerenciar a execução de objetos `Runnable`.*

## Usando a classe `Executors` para obter um `ExecutorService`

A interface `ExecutorService` (do pacote `java.util.concurrent`) *estende* `Executor` e declara vários métodos para gerenciar o ciclo de vida de um objeto `Executor`. Um objeto que implementa a interface `ExecutorService` pode ser criado com os métodos estáticos declarados na classe `Executors` (do pacote `java.util.concurrent`). Usamos a interface `ExecutorService` e um método da classe `Executors` em nosso exemplo, o qual executa três tarefas.

## Implementando a interface `Runnable`

A classe `PrintTask` (Fig. J.12) implementa `Runnable` (linha 5), *para que várias tarefas `PrintTask` possam ser executadas concomitantemente*. A variável `sleepTime` (linha 7) armazena um valor inteiro aleatório de 0 a 5 segundos, criado no construtor de `PrintTask` (linha 17). Cada thread que executa uma `PrintTask` dorme (*sleeps*) pela quantidade de tempo especificada por `sleepTime` e, então, gera na saída o nome da tarefa e uma mensagem indicando que acordou.

```

1 // Fig. J.12: PrintTask.java
2 // A classe PrintTask dorme por um tempo aleatório de 0 a 5 segundos
3 import java.util.Random;
4
5 public class PrintTask implements Runnable
6 {
7 private final int sleepTime; // tempo de dormência aleatório para a thread
8 private final String taskName; // nome da tarefa
9 private final static Random generator = new Random();
10
11 // construtor
12 public PrintTask(String name)
13 {
14 taskName = name; // configura nome da tarefa
15
16 // seleciona tempo de dormência aleatório entre 0 e 5 segundos
17 sleepTime = generator.nextInt(5000); // milissegundos
18 } // fim do construtor de PrintTask
19
20 // o método run contém o código que uma thread executará
21 public void run()
22 {
23 try // coloca a thread para dormir por um sleepTime de tempo

```

**Figura J.12** A classe `PrintTask` dorme por um tempo aleatório de 0 a 5 segundos.

```

24 {
25 System.out.printf("%s going to sleep for %d milliseconds.\n",
26 taskName, sleepTime);
27 Thread.sleep(sleepTime); // coloca a thread para dormir
28 } // fim do try
29 catch (InterruptedException exception)
30 {
31 System.out.printf("%s %s\n", taskName,
32 "terminated prematurely due to interruption");
33 } // fim do catch
34
35 // imprime o nome da tarefa
36 System.out.printf("%s done sleeping\n", taskName);
37 } // fim do método run
38 } // fim da classe PrintTask

```

**Figura J.12** A classe `PrintTask` dorme por um tempo aleatório de 0 a 5 segundos.

Uma `PrintTask` é executada quando uma thread chama o método `run` de `PrintTask`. As linhas 25 e 26 exibem uma mensagem indicando o nome da tarefa em execução no momento e o da tarefa que vai dormir por `sleepTime` milissegundos. A linha 27 chama o método estático `sleep` da classe `Thread` para colocar a thread no estado de *espera agendada* pela quantidade de tempo especificada. Nesse ponto, a thread perde o processador e o sistema permite que outra thread seja executada. Quando a thread acorda, entra novamente no estado *executável*. Quando a `PrintTask` é novamente atribuída a um processador, a linha 36 gera uma mensagem indicando que a tarefa despertou (*done sleeping*) e, então, o método `run` termina. O bloco `catch` nas linhas 29 a 33 é obrigatório, pois o método `sleep` pode lançar uma exceção *verificada* de tipo `InterruptedException`, caso o método `interrupt` de uma thread dormiente seja chamado.

### Usando `ExecutorService` para gerenciar threads que executam objetos `PrintTask`

A Figura J.13 usa um objeto `ExecutorService` para gerenciar threads que executam objetos `PrintTask` (conforme definido na Fig. J.12). As linhas 11 a 13 criam e nomeiam três objetos `PrintTask` para execução. A linha 18 usa o método `newCachedThreadPool` de `Executors` para obter um objeto `ExecutorService` capaz de criar novas threads, à medida que o aplicativo necessitar. Essas threads são usadas pelo objeto `ExecutorService` (`threadExecutor`) para executar os objetos `Runnable`.

```

1 // Fig. J.13: TaskExecutor.java
2 // Usando um objeto ExecutorService para executar objetos Runnable.
3 import java.util.concurrent.Executors;
4 import java.util.concurrent.ExecutorService;
5
6 public class TaskExecutor
7 {
8 public static void main(String[] args)
9 {
10 // cria e nomeia cada runnable
11 PrintTask task1 = new PrintTask("task1");
12 PrintTask task2 = new PrintTask("task2");
13 PrintTask task3 = new PrintTask("task3");
14
15 System.out.println("Starting Executor");

```

**Figura J.13** Usando um objeto `ExecutorService` para executar objetos `Runnable`. (continua)

```

16
17 // cria ExecutorService para gerenciar threads
18 ExecutorService threadExecutor = Executors.newCachedThreadPool();
19
20 // inicia as threads e coloca no estado runnable
21 threadExecutor.execute(task1); // inicia task1
22 threadExecutor.execute(task2); // inicia task2
23 threadExecutor.execute(task3); // inicia task3
24
25 // suspende threads de trabalho quando suas tarefas terminam
26 threadExecutor.shutdown();
27
28 System.out.println("Tasks started, main ends.\n");
29 } // fim de main
30 } // fim da classe TaskExecutor

```

```

Starting Executor
Tasks started, main ends

task1 going to sleep for 4806 milliseconds
task2 going to sleep for 2513 milliseconds
task3 going to sleep for 1132 milliseconds
task3 done sleeping
task2 done sleeping
task1 done sleeping

```

```

Starting Executor
task1 going to sleep for 3161 milliseconds.
task3 going to sleep for 532 milliseconds.
task2 going to sleep for 3440 milliseconds.
Tasks started, main ends.

task3 done sleeping
task1 done sleeping
task2 done sleeping

```

**Figura J.13** Usando um objeto `ExecutorService` para executar objetos `Runnable`.

As linhas 21 a 23 chamam cada uma o método `execute` de `ExecutorService`, o qual executa o objeto `Runnable` passado como argumento (neste caso, uma `PrintTask`) algum tempo no futuro. A tarefa especificada pode ser executada em uma das threads do pool de threads de `ExecutorService`, em uma nova thread criada para executá-la ou na thread que chamou o método `execute` – o `ExecutorService` gerencia esses detalhes. O método `execute` retorna imediatamente de cada chamada – o programa *não* espera que cada `PrintTask` termine. A linha 26 chama o método **`shutdown`** de `ExecutorService`, o qual notifica o `ExecutorService` para que *pare de aceitar novas tarefas, mas continue a executar as que já foram enviadas*. Quando todos os objetos `Runnable` enviados anteriormente tiverem terminado, o `threadExecutor` termina. A linha 28 gera uma mensagem indicando que as tarefas foram iniciadas e que a thread de `main` está finalizando sua execução.

O código em `main` é executado na **thread principal**, uma thread criada pela JVM. O código no método `run` de `PrintTask` (linhas 21 a 37 da Fig. J.12) é executado quando o objeto `Executor` inicia cada `PrintTask` – novamente, isso se dá algum tempo depois que elas são passadas para o método `execute` de `ExecutorService` (Fig. J.13, linhas 21 a 23). Quando `main` termina, o programa continua a ser executado, pois ainda existem tarefas que devem terminar de ser executadas. O programa não terminará até que essas tarefas sejam concluídas.

Os exemplos de saída mostram o nome de cada tarefa e o tempo de dormência da thread. A que tem o menor tempo de dormência *normalmente* acorda primeiro, indica que acordou e termina. Na primeira saída, a thread `main` termina *antes* que qualquer uma das `PrintTasks` gere seu nome e tempo de dormência. Isso mostra que a thread `main` é executada até o fim, antes que as `PrintTasks` tenham uma chance de ser executadas. Na segunda saída, todas as `PrintTasks` geram seus nomes e tempos de dormência *antes* que a thread `main` termine. Além disso, observe, no segundo exemplo de saída, que `task3` dorme antes de `task2`, apesar de termos passado `task2` para o método `execute` de `ExecutorService` antes de `task3`. Isso ilustra o fato de que *não podemos prever a ordem em que as tarefas começarão a ser executadas, mesmo sabendo a ordem em que foram criadas e iniciadas*.

## J.15 Visão geral da sincronização de threads

Quando várias threads compartilham um objeto e ele é modificado por uma ou mais delas, podem ocorrer resultados indeterminados, a não ser que o acesso ao objeto compartilhado seja gerenciado corretamente. Se uma thread está no processo de atualização de um objeto compartilhado e outra thread também tenta atualizá-lo, não está claro qual thread efetivará a atualização. Quando isso acontece, o comportamento do programa não é confiável – às vezes, o programa produzirá os resultados corretos e às vezes não. Em um ou outro caso, não haverá indicação de que o objeto compartilhado foi manipulado incorretamente.

O problema pode ser resolvido fornecendo-se a apenas uma thread por vez um tempo de *acesso exclusivo* ao código que manipula o objeto compartilhado. Durante esse tempo, outras threads que queiram manipular o objeto são mantidas no estado de espera. Quando a thread com acesso exclusivo ao objeto acaba de manipulá-lo, uma das threads que estavam esperando pode prosseguir. Esse processo, chamado de **sincronização de threads**, coordena o acesso a dados compartilhados por várias threads concorrentes. Sincronizando threads dessa maneira, você garante que cada thread que esteja acessando um objeto compartilhado impeça todas as outras threads de fazer isso simultaneamente – isso se chama **exclusão mútua**.

### Monitores

Uma maneira comum de fazer a sincronização é usar **monitores** internos da linguagem Java. Todo objeto tem um monitor e um **bloqueio de monitor** (ou **bloqueio intrínseco**). O monitor garante que o bloqueio de monitor de seu objeto seja mantido no máximo apenas por uma thread por vez e, assim, possa ser usado para impor a exclusão mútua. Se uma operação exige que a thread em execução mantenha um bloqueio enquanto é efetuada, uma thread deve adquirir o bloqueio antes de prosseguir com a operação. Outras threads que estejam tentando efetuar uma operação que exija o mesmo bloqueio serão *obstruídas* até que a primeira thread libere o bloqueio, ponto em que as threads *obstruídas* podem tentar adquirir o bloqueio e prosseguir com a operação.

Para especificar que uma thread deve manter um bloqueio de monitor para executar um bloco de código, o código deve ser colocado em uma **instrução synchronized**. Diz-se que tal código é **protegido** pelo bloqueio de monitor; uma thread deve **adquirir o bloqueio** para executar as instruções protegidas. O monitor só permite que uma thread por vez execute instruções dentro de instruções `synchronized` que bloqueiam o

mesmo objeto, pois apenas uma thread por vez pode manter o bloqueio de monitor. As instruções `synchronized` são declaradas com a **palavra-chave** `synchronized`:

```
synchronized (objeto)
{
 instruções
} // fim da instrução sincronizada
```

onde *objeto* é o objeto cujo bloqueio de monitor vai ser adquirido; normalmente, *objeto* é `this`, caso esse seja o objeto no qual a instrução `synchronized` aparece. Se várias instruções `synchronized` estão tentando ser executadas em um objeto ao mesmo tempo, apenas uma delas pode estar ativa no objeto – todas as outras threads que estão tentando entrar em uma instrução `synchronized` no mesmo objeto têm sua execução temporariamente *obstruída*.

Quando uma instrução `synchronized` termina de ser executada, o bloqueio de monitor do objeto é liberado e uma das threads *obstruídas* que estava tentando entrar em uma instrução `synchronized` pode adquirir o bloqueio para prosseguir. A linguagem Java também permite **métodos** `synchronized`. Antes de ser executado, um método `synchronized` não estático deve adquirir o bloqueio no objeto utilizado para chamá-lo. Do mesmo modo, um método `synchronized` estático deve adquirir o bloqueio na classe utilizada para chamá-lo.

## J.16 Visão geral das coleções concorrentes

Anteriormente neste apêndice, apresentamos várias coleções da API Java Collections. As coleções do pacote `java.util.concurrent` são projetadas e otimizadas especificamente para uso em programas que compartilham coleções entre várias threads. Para obter informações sobre as muitas coleções concorrentes do pacote `java.util.concurrent`, visite

```
docs.oracle.com/javase/6/docs/api/java/util/concurrent/
package-summary.html
```

## J.17 Multithread com interface gráfica do usuário

Os aplicativos Swing apresentam um conjunto de desafios único para a programação multithread. Todos os aplicativos Swing têm uma **thread de despacho de eventos** para tratar das interações com os componentes da interface gráfica do usuário. Interações típicas incluem *atualizar componentes da interface gráfica do usuário* ou *processar ações do usuário*, como cliques de mouse. Todas as tarefas que exigem interação com a interface gráfica do usuário de um aplicativo são colocadas em uma *fila de eventos* e executadas sequencialmente pela thread de despacho de eventos.

*Os componentes da interface gráfica do usuário Swing não têm thread segura* – eles não podem ser manipulados por várias threads sem correrem o risco de obter resultados incorretos. A segurança das threads em aplicativos com interface gráfica do usuário é obtida não pela sincronização das ações de thread, mas *garantindo que os componentes Swing sejam acessados a partir da thread de despacho de eventos* – uma técnica chamada **confinamento de thread**.

Normalmente, é suficiente executar tarefas simples na thread de despacho de eventos em sequência, com manipulações de componentes da interface gráfica do usuário. Se uma tarefa prolongada é executada na thread de despacho de eventos, esta não pode atender às outras tarefas da fila de eventos enquanto estiver ocupada com essa

tarefa. Isso faz que a interface gráfica do usuário deixe de responder. *As tarefas de execução longa devem ser tratadas em threads separadas*, liberando a thread de despacho de eventos para continuar a gerenciar outras interações da interface gráfica do usuário. Evidentemente, para atualizar a interface gráfica do usuário com base nos resultados da tarefa, você deve usar a thread de despacho de eventos, em vez da thread de trabalho que realizou a computação.

### Classe `SwingWorker`

A classe `SwingWorker` (do pacote `javax.swing`) executa tarefas de execução longa em uma thread de trabalho e atualiza componentes Swing a partir da thread de despacho de eventos com base nos resultados da tarefa. A classe `SwingWorker` implementa a interface `Runnable`, significando que *um objeto `SwingWorker` pode ter sua execução agendada em uma thread separada*. A classe `SwingWorker` fornece vários métodos para simplificar a execução de tarefas em uma thread de trabalho e tornar os resultados disponíveis para exibição em uma interface gráfica do usuário. Alguns métodos comuns de `SwingWorker` estão descritos na Fig. J.14. A classe `SwingWorker` é semelhante à classe `AsyncTask`, a qual é frequentemente usada em aplicativos Android.

| Método                      | Descrição                                                                                                                                                                                  |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>doInBackground</code> | Define uma tarefa longa e é chamado em uma thread de trabalho.                                                                                                                             |
| <code>done</code>           | Executado na thread de despacho de eventos quando <code>doInBackground</code> retorna.                                                                                                     |
| <code>execute</code>        | Agenda a execução do objeto <code>SwingWorker</code> em uma thread de trabalho.                                                                                                            |
| <code>get</code>            | Espera pelo término da tarefa e, então, retorna o resultado da tarefa (isto é, o valor de retorno de <code>doInBackground</code> ).                                                        |
| <code>publish</code>        | Envia resultados intermediários do método <code>doInBackground</code> para o método <code>process</code> , para processamento na thread de despacho de eventos.                            |
| <code>process</code>        | Recebe resultados intermediários do método <code>publish</code> e processa esses resultados na thread de despacho de eventos.                                                              |
| <code>setProgress</code>    | Configura a propriedade <code>progress</code> para notificar a quaisquer receptores de alteração de propriedade na thread de despacho de eventos sobre atualizações da barra de andamento. |

**Figura J.14** Métodos de `SwingWorker` comumente usados.

### Executando tarefas em uma thread de trabalho

No próximo exemplo, o usuário digita um número  $n$  e o programa obtém o  $n$ -ésimo número de Fibonacci, o qual calculamos usando um algoritmo recursivo. Para valores grandes, o algoritmo é demorado, de modo que usamos um objeto `SwingWorker` para efetuar o cálculo em uma thread de trabalho. A interface gráfica também permite que o usuário obtenha o próximo número de Fibonacci na sequência a cada clique em um botão, começando com `fibonacci(1)`. Esse breve cálculo é efetuado diretamente na thread de despacho de eventos. O programa pode produzir até o 92º número de Fibonacci – os valores subsequentes estão fora do intervalo que pode ser representado por um `long`. Você pode usar a classe `BigInteger` para representar valores inteiros arbitrariamente grandes.

A classe `BackgroundCalculator` (Fig. J.15) efetua o cálculo de Fibonacci recursivo em uma *thread de trabalho*. Essa classe estende `SwingWorker` (linha 8), sobrescrevendo os métodos `doInBackground` e `done`. O método `doInBackground` (linhas 21 a 24) calcula o

$n$ -ésimo número de Fibonacci em uma thread de trabalho e retorna o resultado. O método `done` (linhas 27 a 43) exibe o resultado em um componente `JLabel`.

```

1 // Fig. J.15: BackgroundCalculator.java
2 // Subclasse de SwingWorker para calcular números de Fibonacci
3 // em uma thread de trabalho.
4 import javax.swing.SwingWorker;
5 import javax.swing.JLabel;
6 import java.util.concurrent.ExecutionException;
7
8 public class BackgroundCalculator extends SwingWorker< Long, Object >
9 {
10 private final int n; // número de Fibonacci a calcular
11 private final JLabel resultJLabel; // componente JLabel para exibir o resultado
12
13 // construtor
14 public BackgroundCalculator(int number, JLabel label)
15 {
16 n = number;
17 resultJLabel = label;
18 } // fim do construtor de BackgroundCalculator
19
20 // código de execução longa a ser executado em uma thread de trabalho
21 public Long doInBackground()
22 {
23 return nthFib = fibonacci(n);
24 } // fim do método doInBackground
25
26 // código a executar na thread de despacho de eventos quando doInBackground retornar
27 protected void done()
28 {
29 try
30 {
31 // obtém o resultado de doInBackground e o exibe
32 resultJLabel.setText(get().toString());
33 } // fim do try
34 catch (InterruptedException ex)
35 {
36 resultJLabel.setText("Interrupted while waiting for results.");
37 } // fim do catch
38 catch (ExecutionException ex)
39 {
40 resultJLabel.setText(
41 "Error encountered while performing calculation.");
42 } // fim do catch
43 } // fim do método done
44
45 // método fibonacci recursivo; calcula o n-ésimo número de Fibonacci
46 public long fibonacci(long number)
47 {
48 if (number == 0 || number == 1)
49 return number;
50 else
51 return fibonacci(number - 1) + fibonacci(number - 2);
52 } // fim do método fibonacci
53 } // fim da classe BackgroundCalculator

```

**Figura J.15** Subclasse de `SwingWorker` para calcular números de Fibonacci em uma thread de trabalho.

`SwingWorker` é uma *classe genérica*. Na linha 8, o primeiro parâmetro de tipo é `Long` e o segundo é `Object`. O primeiro parâmetro de tipo indica o tipo retornado pelo método.

do `doInBackground`; o segundo indica o tipo passado entre os métodos `publish` e `process` para tratar os resultados intermediários. Como não usamos `publish` nem `process` neste exemplo, utilizamos simplesmente `Object` como segundo parâmetro de tipo.

Um objeto `BackgroundCalculator` pode ser instanciado a partir de uma classe que controle uma interface gráfica do usuário. Ele mantém variáveis de instância para um valor inteiro que representa o número de Fibonacci a ser calculado e um componente `JLabel` que exibe os resultados do cálculo (linhas 10 e 11). O construtor de `BackgroundCalculator` (linhas 14 a 18) inicializa essas variáveis de instância com os argumentos passados para o construtor.



### Observação sobre engenharia de software J.3

*Os componentes de interface gráfica do usuário que vão ser manipulados por métodos `SwingWorker`, como os que vão ser atualizados pelos métodos `process` ou `done`, devem ser passados para o construtor da subclasse `SwingWorker` e armazenados no objeto da subclasse. Isso proporciona a esses métodos o acesso aos componentes da interface que eles vão manipular.*

Quando o método `execute` é chamado em um objeto `BackgroundCalculator`, a execução do objeto é agendada em uma thread de trabalho. O método `doInBackground` é chamado a partir da thread de trabalho e ativa o método `fibonacci` (linhas 46 a 52), passando a variável de instância `n` como argumento (linha 23). O método `fibonacci` utiliza recursão para calcular o valor Fibonacci de `n`. Quando `fibonacci` retorna, o método `doInBackground` retorna o resultado.

Após `doInBackground` retornar, o método `done` é chamado automaticamente a partir da thread de despacho de eventos. Esse método tenta configurar o elemento `JLabel` resultante com o valor de retorno de `doInBackground`, chamando o método `get` para recuperar esse valor de retorno (linha 32). Se necessário, o método `get` espera que o resultado esteja pronto, mas como o chamamos a partir do método `done`, o cálculo terminará antes que `get` seja chamado. As linhas 34 a 37 capturam a exceção `InterruptedException`, caso a thread atual seja interrompida enquanto espera pelo retorno de `get`. Essa exceção não ocorrerá neste exemplo, pois o cálculo já terá terminado quando `get` for chamado. As linhas 38 a 42 capturam a exceção `ExecutionException`, a qual é lançada se ocorre uma exceção durante o cálculo.

### Classe `FibonacciNumbers`

A classe `FibonacciNumbers` (Fig. J.16) exibe uma janela contendo dois conjuntos de componentes de interface gráfica do usuário – um para calcular um número de Fibonacci em uma thread de trabalho e outro para obter o próximo número de Fibonacci em resposta ao clique do usuário em um componente `JButton`. O construtor (linhas 38 a 109) coloca esses componentes em elementos `JPanel` intitulados separadamente. As linhas 46 e 47 e as linhas 78 e 79 adicionam dois elementos `JLabel`, um `JTextField` e um `JButton` no `workerJPanel` para permitir que o usuário digite um valor inteiro cujo número de Fibonacci será calculado por `BackgroundWorker`. As linhas 84 e 85 e a linha 103 adicionam dois elementos `JLabel` e um `JButton` ao painel da thread de despacho de eventos para permitir que o usuário obtenha o próximo número de Fibonacci da sequência. As variáveis de instância `n1` e `n2` contêm os dois números de Fibonacci anteriores da sequência e são inicializados com 0 e 1, respectivamente (linhas 29 e 30). A variável de instância `count` armazena o número da sequência calculado mais recentemente e é inicializada com 1 (linha 31). Os dois componentes `JLabel` exibem inicialmente `count` e `n2`, de modo que o usuário verá o texto Fibonacci of 1: 1 no `eventThreadJPanel` quando a interface gráfica começar.



```
1 // Fig. J.16: FibonacciNumbers.java
2 // Usando SwingWorker para efetuar um cálculo longo com os
3 // resultados exibidos em uma interface gráfica do usuário.
4 import java.awt.GridLayout;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
7 import javax.swing.JButton;
8 import javax.swing.JFrame;
9 import javax.swing.JPanel;
10 import javax.swing.JLabel;
11 import javax.swing.JTextField;
12 import javax.swing.border.TitledBorder;
13 import javax.swing.border.LineBorder;
14 import java.awt.Color;
15 import java.util.concurrent.ExecutionException;
16
17 public class FibonacciNumbers extends JFrame
18 {
19 // componentes para calcular o número de Fibonacci de um valor digitado pelo usuário
20 private final JPanel workerJPanel =
21 new JPanel(new GridLayout(2, 2, 5, 5));
22 private final JTextField numberJTextField = new JTextField();
23 private final JButton goJButton = new JButton("Go");
24 private final JLabel fibonacciJLabel = new JLabel();
25
26 // componentes e variáveis para obter o próximo número de Fibonacci
27 private final JPanel eventThreadJPanel =
28 new JPanel(new GridLayout(2, 2, 5, 5));
29 private long n1 = 0; // inicializa com o primeiro número de Fibonacci
30 private long n2 = 1; // inicializa com o segundo número de Fibonacci
31 private int count = 1; // número de Fibonacci atual para exibir
32 private final JLabel nJLabel = new JLabel("Fibonacci of 1: ");
33 private final JLabel nFibonacciJLabel =
34 new JLabel(String.valueOf(n2));
35 private final JButton nextNumberJButton = new JButton("Next Number");
36
37 // construtor
38 public FibonacciNumbers()
39 {
40 super("Fibonacci Numbers");
41 setLayout(new GridLayout(2, 1, 10, 10));
42
43 // adiciona componentes de interface gráfica do usuário ao painel SwingWorker
44 workerJPanel.setBorder(new TitledBorder(
45 new LineBorder(Color.BLACK), "With SwingWorker"));
46 workerJPanel.add(new JLabel("Get Fibonacci of:"));
47 workerJPanel.add(numberJTextField);
48 goJButton.addActionListener(
49 new ActionListener()
50 {
51 public void actionPerformed(ActionEvent event)
52 {
53 int n;
54
55 try
56 {
57 // recupera a entrada do usuário como um valor inteiro
58 n = Integer.parseInt(numberJTextField.getText());
59 } // fim do try
```

---

**Figura J.16** Usando SwingWorker para efetuar um cálculo longo com os resultados exibidos em uma interface gráfica do usuário.

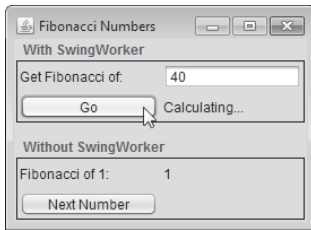
```

60 catch(NumberFormatException ex)
61 {
62 // exibe uma mensagem de erro se o usuário não
63 // digitou um valor inteiro
64 fibonacciJLabel.setText("Enter an integer.");
65 return;
66 } // fim do catch
67
68 // indica que o cálculo começou
69 fibonacciJLabel.setText("Calculating...");
70
71 // cria uma tarefa para efetuar o cálculo em segundo plano
72 BackgroundCalculator task =
73 new BackgroundCalculator(n, fibonacciJLabel);
74 task.execute(); // executa a tarefa
75 } // fim do método actionPerformed
76 } // fim da classe interna anônima
77); // fim da chamada a addActionListener
78 workerJPanel.add(goJButton);
79 workerJPanel.add(fibonacciJLabel);
80
81 // adiciona componentes de interface gráfica do usuário ao painel da
82 // thread de despacho de eventos
83 eventThreadJPanel.setBorder(new TitledBorder(
84 new LineBorder(Color.BLACK), "Without SwingWorker"));
85 eventThreadJPanel.add(nJLabel);
86 eventThreadJPanel.add(nFibonacciJLabel);
87 nextNumberJButton.addActionListener(
88 new ActionListener()
89 {
90 public void actionPerformed((ActionEvent event))
91 {
92 // calcula o número de Fibonacci após n2
93 long temp = n1 + n2;
94 n1 = n2;
95 n2 = temp;
96 ++count;
97
98 // exibe o próximo número de Fibonacci
99 nJLabel.setText("Fibonacci of " + count + ": ");
100 nFibonacciJLabel.setText(String.valueOf(n2));
101 } // fim do método actionPerformed
102 } // fim da classe interna anônima
103); // fim da chamada a addActionListener
104 eventThreadJPanel.add(nextNumberJButton);
105
106 add(workerJPanel);
107 add(eventThreadJPanel);
108 setSize(275, 200);
109 setVisible(true);
110 } // fim do construtor
111
112 // o método main inicia a execução do programa
113 public static void main(String[] args)
114 {
115 FibonacciNumbers application = new FibonacciNumbers();
116 application.setDefaultCloseOperation(EXIT_ON_CLOSE);
117 } // fim de main
118 } // fim da classe FibonacciNumbers

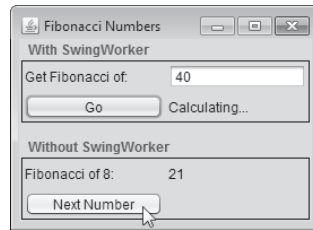
```

**Figura J.16** Usando SwingWorker para efetuar um cálculo longo com os resultados exibidos em uma interface gráfica do usuário. (continua)

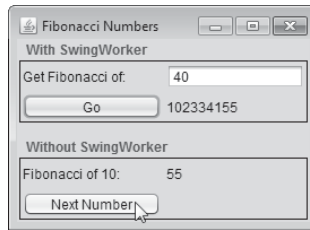
a) Começa a calcular o valor de Fibonacci de 40 em segundo plano



b) Calculando outros valores de Fibonacci enquanto o valor de Fibonacci de 40 continua a ser calculado



c) Fim do cálculo do valor de Fibonacci de 40



**Figura J.16** Usando SwingWorker para efetuar um cálculo longo com os resultados exibidos em uma interface gráfica do usuário.

As linhas 48 a 77 registram a rotina de tratamento de evento para `goJButton`. Se o usuário clica nesse componente `JButton`, a linha 58 obtém o valor digitado no `numberJTextField` e tenta analisá-lo como um inteiro. As linhas 72 e 73 criam um novo objeto `BackgroundCalculator`, passando o valor digitado pelo usuário e o `fibonacciJLabel` utilizado para exibir o resultado do cálculo. A linha 74 chama o método `execute` no `BackgroundCalculator`, agendando sua execução em uma thread de trabalho separada. O método `execute` não espera o `BackgroundCalculator` para terminar de ser executado. Ele retorna imediatamente, permitindo que a interface gráfica do usuário continue a processar outros eventos enquanto o cálculo é efetuado.

Se o usuário clica no `nextNumberJButton` no `eventThreadJPanel`, a rotina de tratamento de evento registrada nas linhas 86 a 102 é executada. As linhas 92 a 95 somam os dois números de Fibonacci anteriores (armazenados em `n1` e `n2`) para determinar o próximo número da sequência, atualizam `n1` e `n2` com seus novos valores e incrementam `count`. Então, as linhas 98 e 99 atualizam a interface gráfica para exibir o próximo número. O código para esses cálculos está no método `actionPerformed`, de modo que são efetuados na *thread de despacho de eventos*. Manipular cálculos curtos assim na thread de despacho de eventos não faz que a interface gráfica deixe de responder, como no algoritmo recursivo para calcular o valor de Fibonacci de um número grande. Como o cálculo de Fibonacci mais longo é efetuado em uma thread de trabalho separada, usando o `SwingWorker`, é possível obter o próximo número de Fibonacci enquanto o cálculo recursivo ainda está em andamento.

## J.18 Para finalizar

Neste apêndice, você usou as classes `ArrayList` e `LinkedList`, as quais implementam a interface `List`. Também usou vários métodos predefinidos para manipular coleções. Em

seguida, você aprendeu a usar a interface `Set` e a classe `HashSet` para manipular uma coleção não ordenada de valores únicos. Discutimos a interface `SortedSet` e a classe `TreeSet` para manipulação de uma coleção ordenada de valores únicos. Então, você aprendeu sobre as interfaces e classes da linguagem Java para manipular pares chave/valor – `Map`, `SortedMap`, `HashMap` e `TreeMap`. Discutimos os métodos estáticos da classe `Collections` para obtenção de modos de exibição não modificáveis e sincronizados de coleções.

Depois, apresentamos os conceitos fundamentais do processamento de arquivos e de fluxos, e estudamos a serialização de objetos. Por fim, apresentamos o `multithread`. Você aprendeu que o Java torna a concorrência acessível por meio da linguagem e de APIs. Aprendeu também que a própria JVM cria threads para executar um programa e que ela também pode criar threads para realizar tarefas de limpeza, como a coleta de lixo. Apresentamos a interface `Runnable`, a qual é usada para especificar uma tarefa que pode ser executada de forma concorrente com outras tarefas. Mostramos como usar a interface `Executor` para gerenciar a execução de objetos `Runnable` por meio de pools de thread, os quais podem reutilizar threads já existentes para eliminar a sobrecarga da criação de uma nova thread para cada tarefa e aumentar o desempenho, otimizando o número de threads para garantir que o processador permaneça ocupado. Discutimos como usar um bloco `synchronized` para coordenar o acesso a dados compartilhados por várias threads concorrentes.

Discutimos também o fato de as interfaces gráficas do usuário Swing não terem thread segura, de modo que todas as interações e modificações feitas em uma interface gráfica devem ser realizadas na thread de despacho de eventos. Examinamos ainda os problemas associados à execução de cálculos longos na thread de despacho de eventos. Então, mostramos como é possível usar a classe `SwingWorker` para efetuar cálculos de execução longa em threads de trabalho e como exibir os resultados de um `SwingWorker` em uma interface gráfica do usuário quando o cálculo termina.

## Exercícios de revisão

**J.1** Preencha os espaços em branco em cada um dos seguintes enunciados:

- Um(a) \_\_\_\_\_ é usado(a) para iterar por uma coleção e pode remover elementos da coleção durante a iteração.
- Um elemento de um objeto `List` pode ser acessado por meio de seu \_\_\_\_\_.
- Supondo que `myArray` contém referências para objetos `Double`, o \_\_\_\_\_ ocorre quando a instrução "`myArray[ 0 ] = 1.25;`" é executada.
- As classes Java \_\_\_\_\_ e \_\_\_\_\_ fornecem os recursos de estruturas de dados do tipo array que podem se redimensionar dinamicamente.
- Supondo que `myArray` contém referências para objetos `Double`, o \_\_\_\_\_ ocorre quando a instrução "`double number = myArray[ 0 ];`" é executada.
- O método \_\_\_\_\_ de `ExecutorService` finaliza cada thread em um `ExecutorService` assim que acaba de executar seu objeto `Runnable` atual, se houver.
- A palavra-chave \_\_\_\_\_ indica que somente uma thread por vez deve ser executada em um objeto.

**J.2** Determine se cada afirmativa é *verdadeira* ou *falsa*. Se for *falsa*, explique o motivo.

- Os valores de tipos primitivos podem ser armazenados diretamente em uma coleção.
- Um objeto `Set` pode conter valores duplicados.
- Um objeto `Map` pode conter chaves duplicadas.
- Um objeto `LinkedList` pode conter valores duplicados.

- e) `Collections` é uma interface.
- f) Objetos `Iterator` podem remover elementos.
- g) O método `exists` da classe `File` retorna `true` se o nome especificado como argumento para o construtor de `File` é um arquivo ou diretório no caminho especificado.
- h) Arquivos binários podem ser lidos por humanos em um editor de texto.
- i) Um caminho absoluto contém todos os diretórios, começando no diretório-raiz, que levam a um arquivo ou diretório específico.

## Respostas dos exercícios de revisão

- J.1** a) `Iterator`. b) índice. c) autoboxing. d) `ArrayList`, `Vector`. e) auto-unboxing. f) shutdown. g) `synchronized`.
- J.2** a) Falsa. O autoboxing ocorre ao se adicionar um tipo primitivo a uma coleção. Isso significa que o tipo primitivo é convertido para sua correspondente classe encapsuladora de tipos. b) Falsa. Um objeto `Set` não pode conter valores duplicados. c) Falsa. Um objeto `Map` não pode conter chaves duplicadas. d) Verdadeira. e) Falsa. `Collections` é uma classe; `Collection` é uma interface. f) Verdadeira. g) Verdadeira. h) Falsa. Arquivos de texto podem ser lidos por humanos em um editor de texto. Os arquivos binários também podem, mas somente se os bytes deles representarem caracteres ASCII. i) Verdadeira.

## Exercícios

- J.3** Defina cada um dos termos a seguir:
- a) `Collection`
  - b) `Collections`
  - c) `Comparator`
  - d) `List`
  - e) `HashMap`
  - f) `ObjectOutputStream`
  - g) `File`
  - h) `ObjectOutputStream`
  - i) fluxo baseado em bytes
  - j) fluxo baseado em caracteres
- J.4** Responda sucintamente às seguintes perguntas:
- a) Qual é a principal diferença entre um objeto `Set` e um objeto `Map`?
  - b) O que acontece quando você adiciona um valor de tipo primitivo (por exemplo, `double`) a uma coleção?
  - c) Você pode imprimir todos os elementos de uma coleção sem usar um objeto `Iterator`? Em caso positivo, como?
- J.5** (*Eliminação de duplicatas*) Escreva um programa que leia uma série de nomes e elimine as duplicatas, armazenando-as em um objeto `Set`. Permita que o usuário procure um nome.
- J.6** (*Contando letras*) Modifique o programa da Fig. J.9 para contar o número de ocorrências de cada letra, em vez de cada palavra. Por exemplo, a string "HELLO THERE" contém dois H, três E, dois L, um O, um T e um R. Exiba os resultados.
- J.7** (*Seletor de cor*) Use um objeto `HashMap` a fim de criar uma classe reutilizável para escolher uma das 13 cores predefinidas na classe `Color`. Os nomes das cores devem ser usados como chaves, e os objetos `Color` predefinidos devem ser usados como valores. Coloque essa classe em um pacote que possa ser importado para qualquer programa Java. Use sua nova

classe em um aplicativo que permita ao usuário selecionar uma cor e desenhar uma forma com essa cor.

- J.8** (*Contando palavras duplicadas*) Escreva um programa que determine e imprima o número de palavras duplicadas em uma frase. Trate as letras maiúsculas e minúsculas como iguais. Ignore a pontuação.
- J.9** (*Números primos e fatores primos*) Escreva um programa que receba um número inteiro como entrada do usuário e determine se ele é primo. Se o número não for primo, exiba seus fatores primos. Lembre-se de que os fatores de um número primo são somente 1 e o próprio número primo. Todo número que não é primo tem um fator primo exclusivo. Por exemplo, considere o número 54. Os fatores primos de 54 são 2, 3, 3 e 3. Quando os valores são multiplicados, o resultado é 54. Para o número 54, a saída dos fatores primos deve ser 2 e 3. Use objetos Set como parte de sua solução.
- J.10** (*Ordenando palavras com um objeto TreeSet*) Escreva um programa que use o método `split` de `String` para transformar em tokens uma linha de entrada de texto digitada pelo usuário e colocar cada token em um objeto `TreeSet`. Imprima os elementos do objeto `TreeSet`. [*Obs.:* isso deve fazer que os elementos sejam impressos em ordem crescente.]
- J.11** (*Bouncing Ball - Bola saltitante*) Escreva um programa que faça uma bola azul saltitar dentro de um elemento `JPanel`. A bola deve começar a se mover com um evento `mousePressed`. Quando ela atingir a margem do elemento `JPanel`, deverá quicar e continuar na direção oposta. A bola deve ser atualizada com um objeto `Runnable`.

# K



## Tabela de precedência de operadores

Os operadores são mostrados em ordem decrescente de precedência, de cima para baixo (Fig. K.1).

| Operador                                 | Descrição                                                                                                                                                              | Associatividade         |
|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| ++<br>--                                 | incremento pós-fixado unário<br>decremento pós-fixado unário                                                                                                           | direita para a esquerda |
| ++<br>--<br>+<br>-<br>!<br>~<br>( tipo ) | incremento prefixado unário<br>decremento prefixado unário<br>mais unário<br>menos unário<br>negação lógica unária<br>complemento bit a bit unário<br>conversão unária |                         |
| *<br>/<br>%                              | multiplicação<br>divisão<br>resto                                                                                                                                      |                         |
| +<br>-                                   | adição ou concatenação de strings<br>subtração                                                                                                                         |                         |
| <<<br>>><br>>>>                          | deslocamento à esquerda<br>deslocamento à direita com sinal<br>deslocamento à direita sem sinal                                                                        |                         |
| <<br><=<br>><br>>=<br>instanceof         | menor que<br>menor ou igual a<br>maior que<br>maior ou igual a<br>comparação de tipo                                                                                   |                         |
| ==<br>!=                                 | é igual a<br>não é igual a                                                                                                                                             |                         |
| &                                        | E bit a bit<br>E lógico booleano                                                                                                                                       | esquerda para a direita |
| ^                                        | OU exclusivo bit a bit<br>OU exclusivo lógico booleano                                                                                                                 |                         |
|                                          | OU inclusivo bit a bit<br>OU inclusivo lógico booleano                                                                                                                 |                         |

**Figura K.1** Tabela de precedência de operadores.

| Operador | Descrição                                               | Associatividade         |
|----------|---------------------------------------------------------|-------------------------|
| &&       | E condicional                                           | esquerda para a direita |
|          | OU condicional                                          | esquerda para a direita |
| ?:       | condicional                                             | direita para a esquerda |
| =        | atribuição                                              | direita para a esquerda |
| +=       | adição e atribuição                                     |                         |
| -=       | subtração e atribuição                                  |                         |
| *=       | multiplicação e atribuição                              |                         |
| /=       | divisão e atribuição                                    |                         |
| %=       | resto e atribuição                                      |                         |
| &=       | E bit a bit e atribuição                                |                         |
| ^=       | OU exclusivo bit a bit e atribuição                     |                         |
| =        | OU inclusivo bit a bit e atribuição                     |                         |
| <<=      | deslocamento à esquerda bit a bit e atribuição          |                         |
| >>=      | deslocamento à direita bit a bit com sinal e atribuição |                         |
| >>>=     | deslocamento à direita bit a bit sem sinal e atribuição |                         |

**Figura K.1**    Tabela de precedência de operadores.



# L



## Tipos primitivos

| Tipo                                                                                                  | Tamanho em bits | Valores                                                                                                                                                          | Padrão                               |
|-------------------------------------------------------------------------------------------------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------|
| boolean                                                                                               |                 | true ou false                                                                                                                                                    |                                      |
| [Obs.: A representação de um valor booleano é específica da Java Virtual Machine em cada plataforma.] |                 |                                                                                                                                                                  |                                      |
| char                                                                                                  | 16              | '\u0000' a '\uFFFF' (0 a 65535)                                                                                                                                  | (conjunto de caracteres Unicode ISO) |
| byte                                                                                                  | 8               | $-128$ a $+127$ ( $-2^7$ a $2^7 - 1$ )                                                                                                                           |                                      |
| short                                                                                                 | 16              | $-32.768$ a $+32.767$ ( $-2^{15}$ a $2^{15} - 1$ )                                                                                                               |                                      |
| int                                                                                                   | 32              | $-2.147.483.648$ a $+2.147.483.647$ ( $-2^{31}$ a $2^{31} - 1$ )                                                                                                 |                                      |
| long                                                                                                  | 64              | $-9.223.372.036.854.775.808$ a $+9.223.372.036.854.775.807$ ( $-2^{63}$ a $2^{63} - 1$ )                                                                         |                                      |
| float                                                                                                 | 32              | Intervalo negativo:<br>$-3,4028234663852886E+38$ a $-1,40129846432481707E-45$<br>Intervalo positivo:<br>$1,40129846432481707E-45$ a $3,4028234663852886E+38$     | (ponto flutuante IEEE 754)           |
| double                                                                                                | 64              | Intervalo negativo:<br>$-1,7976931348623157E+308$ a $-4,94065645841246544E-324$<br>Intervalo positivo:<br>$4,94065645841246544E-324$ a $1,7976931348623157E+308$ | (ponto flutuante IEEE 754)           |

**Figura L.1** Tipos primitivos da linguagem Java.

Para obter mais informações sobre IEEE 754, visite [grouper.ieee.org/groups/754/](http://grouper.ieee.org/groups/754/).



# Índice

## Símbolos

^, OU exclusivo lógico booleano 410-411, **412**  
tabela-verdade 412-413  
, (vírgula) flag de formatação **402**  
--, pré-decremento/pós-decremento **394-395**  
-, subtração 343, 344  
!, NÃO lógico 410-411, **413-414**  
tabela-verdade 413-414  
!=, não é igual a 346  
?!, operador condicional ternário **381**  
. separador ponto **358**  
Android Developer Tools  
marcadores de régua no editor  
Graphical Layout 56  
{, chave de abertura 333  
}, chave de fechamento 333  
@Override, anotação **530-531**  
\* curinga em um nome de arquivo 359  
\*, multiplicação 343, 344  
\*==, operador de multiplicação e atribuição 395-396  
/, divisão 343, 344  
/\* \*/ comentário tradicional **333**  
//, comentário de fim de linha **333**  
/==, operador de divisão e atribuição 395-396  
\\, sequência de escape com barra invertida 337  
\", sequência de escape com aspas duplas 337  
\\n, sequência de escape com nova linha 337  
\\r, sequência de escape com retorno de carro 337  
\\t, sequência de escape com tabulação horizontal 337  
&, E lógico booleano 410-411, **412**  
&&, E condicional **410-411**, 411-412  
tabela-verdade 411-412  
%, resto 343, 344  
%==, operador de resto e atribuição 395-396  
%d, especificador de formato **341**  
%f, especificador de formato 353, **371**

%s, especificador de formato **338**  
-, flag de formatação (sinal de subtração) **401-402**  
+, adição 343, 344  
++, pré-incremento/pós-incremento **394-395**  
+==, operador de adição e atribuição **394-395**  
<, menor que 347  
<==, menor ou igual a 347  
==, operador de atribuição 341  
-==, operador de subtração e atribuição 395-396  
== para determinar se duas referências que se referem ao mesmo objeto 546  
==, é igual a 346  
>, maior que 347  
>==, maior ou igual a 347  
|, OU inclusivo lógico booleano 410-411, **412-413**  
||, OU condicional 410-411, **411**  
tabela-verdade 412-413

## Numéricos

0, flag **459**  
0, flag de formato 498  
100 Destinations 6

## A

abordagem baseada em aplicativos **2**  
abreviando expressões de atribuição 394-395  
abrindo um banco de dados 298-299  
abrir um arquivo **645-646**  
abs, método de Math 424  
abstract, palavra-chave **552-553**  
Abstract Window Toolkit Event, pacote **431-432**  
AbstractButton, classe **618-619**, 621  
addActionListener, método 621  
setRolloverIcon, método **621**  
ação 380, 383  
ação a executar **378**  
acelerômetro **15**  
detecção 230-231  
acessando provedores de conteúdo  
Android 13

## Acessibilidade

Content Description, propriedade **59-60**, 114  
Explore by Touch **39-40**, **59**  
TalkBack **39-40**, **59**  
TalkBack e adaptação para a localidade 64-65  
acessibilidade ix, 32, 39-40, 59, 112  
modo explorar pelo toque 9-10  
acesso à rede 13  
acesso a serviços Android 119  
acesso ao sistema de arquivos 13  
acesso de pacote **520**  
Acordo de Licença de Usuário Final (EULA) 309, **310**  
ACTION\_SEND, constante da classe Intent **130**  
ACTION\_VIEW, constante da classe Intent **128**  
ActionEvent, classe **613-614**, 614-615, 618-619  
método getActionCommand **614-615**, 621  
ActionListener, interface **613-614**, 617-618  
método actionPerformed 613-614, 616-617  
actionPerformed, método da interface ActionListener 613-614, 616-617  
Activity, classe **71-72**, 85  
enviada para segundo plano 191-192  
findFragmentById, método **144**  
getFragmentManager, método **144-145**, 161-162, 175  
getMenuInflater, método **162-163**  
getResources, método **127**  
getString, método **160-161**  
getString, método com vários argumentos 130  
getSystemService, método 230  
métodos de ciclo de vida 186-187  
onCreate, método **71-72**, 185  
onCreateOptionsMenu, método **143-144**, 161-162  
onDestroy, método 185, **186-187**  
onOptionsItemSelected, método **143-144**, 162-163

- onPause, método 185, **186-187**
- onResume, método 185
- onStart, método **161-162**, 185
- onStop, método 185
- runOnUiThread, método **205-206**
- setContentView, método **87-88**
- setRequestedOrientation, método **160-161**
- setVolumeControlStream, método **186-187**, 190-191
- Activity, templates de 44
- Activity Not Responding (ANR), caixa de diálogo 265-266
- activity\_main.xml **49-50**
- ActivityNotFoundException, classe 107-108
- adaptação aos locais de uma variável na memória do computador 342
- adaptação local 52, 61-62, 149
- Adapter, classe ix, **106-107**, 219-220
- AdapterView, classe ix, **106-107**, **120**
- AdapterView.OnItemClickListener, interface **120**, 281-282
- AdapterView.OnItemLongClickListener, interface **120**
- add, método
  - ArrayList<T> **490**
  - LinkedList<T> **637**
  - List<T> **633**, 635
- add, método da classe
  - FragmentTransaction **275**
- addActionListener, método
  - da classe AbstractButton 621
  - da classe JTextField **613-614**
- addAll, método
  - Collections 638
  - List **635**
- addCallback, método da classe
  - SurfaceHolder **195**
- addFirst, método de LinkedList **637**
- addLast, método de LinkedList **637**
- addPreferencesFromResource, método
  - da classe PreferenceFragment 175
- Address Book, aplicativo ix, 15
- addToBackStack, método da classe
  - FragmentTransaction **276**
- adição 343, 344
- adicionando componentes a uma linha 74-75
- adicionar uma classe a um projeto 189-190
- Adjust View Bounds, propriedade de um ImageView 154
- AdMob 315, 316
- adquirir o bloqueio **653-654**
- ADT (Android Development Tools Plugin) 13-14
- AlertDialog, classe **107-108**, 119, 124, 218
- AlertDialog.Builder, classe **107**, 124
- algoritmo **378**, 383
  - na Java Collections Framework 638
- algoritmo de embaralhamento Fisher-Yates 465-466
- Algoritmo de Euclides 545
- algoritmo do acordar e levantar (rise-and-shine algorithm) 378
- alinhado à esquerda **401-402**
- alinhar a saída à direita **401-402**
- alpha, animação de para um View **157-158**
- alpha, método da classe Color **249**
- altura de uma linha de tabela 74-75
- Amazon Mobile, aplicativo 316
- ambiente de desenvolvimento integrado (IDE) 13-14, 36
- análise **18**
- análise e projeto orientados a objetos (OOAD) **18**
- Analog Clock, exercício do aplicativo 212
- Android (developer.android.com), documentação para desenvolvedor de xiv-xv
- Android (developer.android.com/sdk/installing/studio.html), documentação para desenvolvedor de xiv-xv
- Android, acesso a serviços 119
- Android, código-fonte e documentação do
  - FAQ (perguntas frequentes) 4
  - governance philosophy 4
  - licenses (licenças) 4
  - source cod (código-fonte) 4
- Android, emulador **xxii**, 39
- Android, fabricantes de dispositivos ix
- Android, lojas de aplicativos 322-323
  - Amazon Appstore 322-323
  - AndroidPIT 322-323
  - Appitalism 322-323
  - GetJar 322-323
  - Handango 322-323
  - Moborobo 322-323
  - Mplayit 322-323
  - Opera Mobile Store 322-323
  - Samsung Apps 322-323
  - SlideMe 322-323
- Android, projeto
  - res, pasta **46-47**, **52**
  - value, pasta **52**
- android.app, pacote **71-72**, 85, 106-107, 119, 143-145, 265-266
- android.content, pacote **106-107**, **119**, 219-220
- android.content.res, pacote 146, **160-161**, **168-169**
- android.database, pacote **265-266**
- android.database.sqlite, pacote **265**
- android.graphics, pacote 187, 219
- android.graphics.drawable, pacote 173
- android.media, pacote 186-187
- android.net, pacote **119**
- android.os, pacote 85, 146, **265**
- android.preference, pacote **143-144**
- android.text, pacote **73-74**, 85
- android.util, pacote 148, 194
- android.view, pacote **120**, 143-144, 186-187, 219-220
- android.view.animation, pacote 146
- android.view.inputmethod, pacote **120**
- android.widget, pacote **72-73**, 85, 106-107, 120, 146-147, 265-266
- android:background, atributo de TextView 268-269
- android:duration, atributo de uma animação de translate **158-159**
- android:fromXDelta, atributo de uma animação de translate **157-158**
- android:startOffset, atributo de uma animação de translate **158-159**
- android:toXDelta, atributo de uma animação de translate **157-158**
- Android 2.2 (Froyo) 7
- Android 2.3 (Gingerbread) **8-9**
- Android 3.x
  - Honeycomb **8-9**
- Android 4.0 (Ice Cream Sandwich) **8-9**
- Android Asset Studio 310
- Android Beam **9-10**, 10-11
- Android Cloud to Device Messaging (C2DM) 7
- Android Developer Tools ix
- Android Discuss, grupo de discussão 32-33
- Android Jelly Bean **9-10**
- Android KitKat **10-11**
- Android Lint 40-42, 59-60
- Android Manifest, editor 74-75
- Android Manifest, editor **91-93**
- Android Programming Quiz, exercício do aplicativo 180
- Android Resources, editor 62-63
- Android SDK Manager **xxi**
- Android SDK/ADT Bundle **xix**
- Android SDK/ADT bundle **xx**, **xxi**, 19, 40-41
- Android Studio **3**, 13, **13-14**, 39-40
- Android Support Library 72-73, 143-144, 220-221, **220-221**, 246
- Android Virtual Device (AVD) **xxii**, **13-14**, 19, 23, 58
  - Setting hardware emulation options 29-30

- Android Virtual Device Manager xxii
- Android@Home, framework **9-10**
- AndroidLicenser 322-323
- AndroidManifest.xml **74-75**, 109
- anim, pasta de um projeto Android **48, 146**
- animação com tween 146, **157-158**
- animação de propriedade ix-x, 146, 157-158
- animação de scale para um View **157**
- animação x-xi
  - animação de alpha para um View **157-158**
  - animação de rotate para um View **157-158**
  - animação de scale para um View **157-158**
  - animação de translate para um View **157-158**
  - baseada em View **157-158**
  - framework 8-9
  - manual 186-187
  - opções em um arquivo XML 146-147
  - set **157-158**
  - thread 186-187
  - tween **157-158**
- Animation, classe **146-147**
  - setRepeatCount, método **148**, 168
- AnimationUtils, classe **146**, 168
  - loadAnimation, método **146**, 168
- animator, pasta de um projeto Android **48, 146**
- ANR (Activity Not Responding), caixa de diálogo 265-266
- ANR (Application Not Responding), caixa de diálogo **86**, 122
- anti-aliasing 236-237
- anúncio incorporado ao aplicativo 314-315, **316**
- anúncios móveis 315
- aparência e comportamento Nimbus **608**
- API (interface de programação de aplicativos) **339, 422**
- API Calendar 9-10
- API Concurrency 648-649
- API Java 422, 580
- API Social 9-10
- API Text-to-Speech 9-10
- APIs Android 5
- APIs de acessibilidade 9-10
- APIs Google 5
- .apk, arquivo (arquivo de pacote de aplicativo Android) 311
- aplicativo **332**, 333, 357
  - argumentos de linha de comando **425**
- aplicativo de tela única 44
- aplicativo gratuito 314-315
- aplicativo robusto 587
- aplicativo xix
- aplicativos de código-fonte aberto 4
- aplicativos de marca
  - Amazon Mobile 316
  - Bank of America 316
  - Best Buy 316
  - CNN 316
  - Epicurious Recipe 316
  - ESPN ScoreCenter 316
  - NFL Mobile 316
  - NYTimes 316
  - Pocket Agent 316
  - Progressive Insurance 316
  - UPS Mobile 316
  - USA Today 316
  - Wells Fargo Mobile 316
  - Women's Health Workouts Lite 316
- aplicativos do livro ix
- aplicativos habilitados para a Internet ix-x
- aplicativos móveis HTML5 ix-x
- Application Not Responding (ANR), caixa de diálogo **86**, 122
- Application Programming Interface Java (API Java) **339**, 422, 430-431
- apply, método da classe SharedPreferences.Editor **126**
- área cliente 38, 106-107
- área de trabalho **19**
- área de um círculo 545
- ARGB, esquema de cores 24-25
- argb, método da classe Color **249**
- ARGB **247**
- ARGB\_8888-88, constante **238-239**
- argumento de linha de comando **425**
- argumento para um método **334**, 359
- ArithmeticException, classe **588**, 594
- arquivo **644-645**
- arquivo binário **644-645**
- arquivo de código 320
- arquivo de manifesto 309, 320
- arquivo de texto **644-645**
- arquivos de mídia 185
- arquivos de som 189-190
- arrastar a caixa de rolagem 623-624
- array **452**, 644-645
  - ignorando o elemento zero 461
  - passar um array para um método 468-469
  - passar um elemento de array para um método 468-469
  - variável de instância length **453**
  - verificação de limites **460**
- array bidimensional **476-477**, 478
- array bidimensional com três linhas e quatro colunas 476-477
- array de apoio **635-636**
- array de arrays unidimensionais 477
- array de  $m$  por  $n$  **476-477**
- array multidimensional 476-477
- array redimensionável
  - implementação de uma List 631
- ArrayAdapter, classe **106-107**, **120**, 123, 265-266
- arraycopy, método da classe System **485-486**, 487-488
- ArrayList, classe 106-107, 119, **148**
- ArrayList<T>, classe genérica **488**, **631**
  - add, método **490**
  - clear, método 488
  - contains, método 488, **490**
  - get, método **490**
  - indexOf, método 488
  - remove, método 488, **490**
  - size, método **490**
  - trimToSize, método 488
- Arrays, classe **485-486**
  - asList, método **635-636**, 635-636
  - binarySearch, método **485-486**
  - equals, método **485-486**
  - fill, método **485-486**
  - sort, método **485-486**
- arredondando um número 343, 424, 447-448
- arredondar um número de ponto flutuante para propósitos de exibição 391-392
- árvore 640-641
- asList, método de Arrays **635**, 635
- aspas duplas, " 334, 337
- AssetManager, classe **146**
  - método list 169-170
- assets, pasta de um aplicativo Android **144-145**
- assinando aplicativos 309
- assinar digitalmente seu aplicativo 312-313
- assinatura 442-443
- assinatura de um método **442-443**
- associatividade de operadores **344**, 349
  - direita para a esquerda 344
  - esquerda para a direita 349
- associativo
  - direita para a esquerda 390-391
- AsyncTask, classe **265-266**, 282-283, **283-284**, 284-285, 294-297
  - execute, método **282-283**
- atribuindo referências de superclasse e subclasse a variáveis de superclasse e subclasse 550-551
- atribuir um valor a uma variável 341
- atributo
  - de um objeto 18, 36

de uma classe 16  
na UML 18, 36  
AttributeSet, classe 194  
atualizando um banco de dados 298  
áudio x-xi, 13  
AudioManager, classe **186-187**, 195  
autoboxing **630**  
avaliação da esquerda para a direita 345  
avaliação de curto-circuito **412-413**  
AVD (Android Virtual Device) **xxii**, **13-14**, 19, 23  
AWEvent, classe 615-616

## B

Background, propriedade de View 81  
baixando código-fonte xiii-xiv  
banco de dados  
abrindo 298-299  
atualizando 298-299  
criando 298-299  
número da versão 302-303  
Bank of America, aplicativo 316  
barra de ação 44, 138, 139  
barra de aplicativo 22-24  
barra de asteriscos 458, 459  
barra de rolagem de um JComboBox **623**  
barra de sistema 38, 106-107, 264  
barra invertida (\) **337**  
baseado em eventos **609**  
BasePlusCommissionEmployee, classe  
estende CommissionEmployee 562-563  
beginTransaction, método da classe  
FragmentManager **275**  
bens virtuais 317-318  
biblioteca de classes **5**  
Biblioteca de classes Java **339**, **422**  
BigDecimal, classe 370  
BigInteger, classe 654-655  
binarySearch, método  
de Arrays **485-486**, 487-488  
de Collections 638  
bit a bit, operadores 410-411  
Bitmap, classe **187**, **219**, 252  
codificação de bitmap **238-239**  
createBitmap, método **238-239**  
eraseColor, método **253-254**  
Bitmap.Config.ARGB\_8888, constante **238-239**  
Blackjack, exercício do aplicativo 135  
melhorado 136  
Blank Activity, template **44**  
Block Breaker Game, exercício do  
aplicativo 257-258  
melhorado 257-258  
bloco **382**  
bloqueio de monitor **652-653**

bloqueio intrínseco **652-653**  
blue, método da classe Color **249**  
Bluetooth Health Devices 9-10  
Body Mass Index Calculator, exercício  
do aplicativo 96-97  
boolean  
expressão **381**  
promoções 430-431  
Boolean, classe **630**  
boolean, tipo primitivo **381**, 666  
botão **618-619**  
botão de aplicativos recentes 22-24  
botão de comando **618-619**  
botão de opção **618-619**  
botão voltar 22-24  
botões de alternância **618-619**  
botões virtuais em um dispositivo  
Android **22-24**  
Bouncing Ball Game, exercício do  
aplicativo 212  
break, instrução **407-408**, 410-411  
Brick Game, exercício do aplicativo  
212  
Bundle, classe 85, **87**  
método putLong **276**  
para um Intent 131  
Byte, classe **630**  
byte, palavra-chave 666  
byte, tipo primitivo 403-404  
promoções 430-431

## C

C2DM (Android Cloud to Device  
Messaging) 7  
cabeçalho de método **357**  
caixa de combinação 622  
caixa de diálogo modal **107-108**  
caixa de rolagem **623-624**  
caixa de seleção **618-619**  
caixa de texto 72-73  
cálculo aritmético 343  
cálculos 350  
câmera 5  
caminho absoluto **645-646**, 646-647  
caminho relativo **645-646**  
campo **362**  
valor inicial padrão **365**  
campo de texto 72-73  
campo de uma classe 438-439  
campos “ocultos” 438-439  
Cannon Game, aplicativo ix, 15  
Cannon Game, exercício do aplicativo  
211  
canRead, método de File 646-647  
Canvas, classe **187-188**, 219-220  
drawBitmap, método **239-240**  
drawCircle, método **203-204**  
drawLine, método **204-205**  
drawPath, método **239-240**, 244  
drawRect, método **203-204**  
drawText, método **203-204**  
canWrite, método de File 646-647  
captura de tela 312-313  
Capturando exceções com  
Superclasses, exercício 606  
Capturando exceções usando a classe  
Exception, exercício 606  
Capturando exceções usando escopos  
externos, exercício 606  
capturar  
uma exceção de superclasse 596-  
597  
uma exceção 589  
Car Payment Calculator, exercício do  
aplicativo 95-96  
caractere de eco da classe  
JPasswordField **609**  
caractere de escape **337**  
caractere de espaço **333**  
caractere de nova linha **336**  
caractere de tabulação, \t 337  
caractere especial 340  
características de desenho 187-188  
cor 187-188  
espessura da linha 187-188  
tamanho da fonte 187-188  
características de excelentes  
aplicativos 31  
Card Game, exercício do aplicativo 136  
carregar um URL em um navegador  
web 107-108  
case, palavra-chave **407-408**  
cassino 434-435  
catch  
bloco **592**, 594-595, 597-598,  
600-601, 603-604  
cláusula **592**  
palavra-chave **592**  
Catch, bloco **461-462**  
cd para mudar de diretório 335  
ceil de Math, método 424  
Celsius 626-627  
equivalente de uma temperatura  
em Fahrenheit 545  
certificado digital **312-313**  
chamada de método **16-17**, 426  
chamada por referência **470-471**  
chamada por valor **470-471**  
chamadas de método em linha **506**  
chamar um método **366**  
changeCursor, método da classe  
CursorAdapter **284-285**  
char  
palavra-chave 666  
promoções 430-431  
tipo primitivo **340**, 403-404  
Character, classe **630**

- chave criptográfica 309
- chave de abertura, { **333**, 334, 339
- chave de fechamento, } **333**, 334, 339, 390-391
- chave privada **312-313**
- chaves ({ e }) 382, 399-400, 456
  - não exigidas 407-408
- check-in 324-325
- chegada de mensagem de rede 594
- ciclo de vida de fragmento 144-145
- cifrões (\$) 333
- circunferência 353
- Class, classe 547-548, **568**
  - método `getName` 547-548, **568**
- class, palavra-chave **333**, 357
- .class, arquivo **335**
  - separar um para toda classe 500
- classe 13, **16-17**
  - arquivo **335**
  - campo **362**
  - class, palavra-chave 357
  - construtor **358**, 367
  - construtor padrão 367
  - declara um método **356**
  - declaração **333**
  - instanciando um objeto **356**
  - método `get` 501-502
  - método `set` 501-502
  - nome **333**
  - ocultação de dados **363**
  - variável de instância **18**, **362**, 424
- classe abstrata **552**, 553, 571
- classe adaptadora **624-625**
- classe aninhada **612-613**
- classe
  - `ArrayIndexOutOfBoundsException` 460, **461-462**
- classe concreta **552-553**
- classe de nível superior **612-613**
- classe driver* 357
- classe encapsuladora de tipos **630**
- classe genérica **488**
- classe interna anônima 71, 608, 612, **623**
- classes
  - `AbstractButton` **618-619**, 621
  - `ActionEvent` **613**, 614, 618
  - `Activity` **71-72**, 85
  - `ActivityNotFoundException` 107
  - `Adapter` **106-107**
  - `AdapterView` **106-107**, 120
  - `AlertDialog` **107-108**, 119
  - `AlertDialog.Builder` **107-108**
  - `Animation` **146-147**
  - `AnimationUtils` **146-147**, 168-169
  - `ArithmeticException` **588**
  - `ArrayAdapter` **106-107**, 120, 123
  - `ArrayIndexOutOfBoundsException` 460-462
  - `ArrayList` 106-107, 119, **148**
  - `ArrayList<T>` **488**, 488, **490**, **631**, 631
  - `Arrays` **485-486**
  - `AssetManager` **146**
  - `AsyncTask` **265-266**, 282-283, 294
  - `AttributeSet` 194
  - `AudioManager` **186-187**, 195
  - `AWTEvent` 615-616
  - `BigDecimal` 370
  - `BigInteger` 654-655
  - `Bitmap` **187**, **219**, 252
  - `Boolean` **630**
  - `Bundle` 85, 87
  - `Byte` **630**
  - `Canvas` **187-188**, 219-220
  - `Character` **630**
  - `Class` 547-548, **568**
  - `Collections` 119, **148**, **631**
  - `Color` 249
  - `Configuration` **160-161**
  - `ContentResolver` **219-220**
  - `ContentValues` **299-300**
  - `Context` **119**
  - `Cursor` **265-266**
  - `CursorAdapter` **265-266**, 281-282
  - `CursorFactory` 302-303
  - `DialogFragment` **143-144**, 175
  - `DialogInterface` **119**
  - `Display` **146-147**, 161-162
  - `Double` **630**
  - `Drawable` **173**
  - `EditText` **72-73**, 85
  - `EnumSet` **514**
  - `Error` **595**
  - `EventListenerList` **617-618**
  - `Exception` **594-595**
  - `ExecutionException` 657
  - `Executors` **659-650**
  - `File` **645-646**
  - `FileInputStream` **645-646**
  - `FileOutputStream` **645-646**
  - `FileReader` **645-646**
  - `FileWriter` **645-646**
  - `Float` **630**
  - `FlowLayout` 611
  - `Formatter` **645-646**
  - `Fragment` **143-144**
  - `FragmentManager` **144-145**
  - `FragmentTransaction` **144**, **264**, 275, 276
  - `FrameLayout` **188-189**
  - `GestureDetector`.
    - `SimpleGestureListener` 240-241
  - `GestureDetector`.
    - `Simple-OnGestureListener` **219**
  - `GridLayout` **72-73**, **110**
  - `Handler` **146-147**
  - `HashMap` **641-642**
  - `HashSet` **640-641**
  - `Hashtable` **641-642**
  - `ImageButton` **106-107**, 113-114, 120
  - `ImageView` **39-40**, 56
  - `IndexOutOfBoundsException` 461
  - `InputMethodManager` **120**
  - `InputMismatchException` **589**
  - `InputStream` 173
  - `Integer` **630**
  - `Intent` **107-108**, 119
  - `InterruptedException` **650-651**
  - `JButton` **618-619**, 621
  - `JComboBox` **622**
  - `JComponent` 617-618, 622
  - `JPasswordField` **609**, 614-615
  - `JTextComponent` **609**, 612-613
  - `TextField` **609**, 613, 616
  - `KeyEvent` **618-619**
  - `LayoutInflater` **144-145**
  - `LinearLayout` **72-73**
  - `LinkedList` **631**
  - `ListActivity` **106-107**, 119
  - `FragmentManager` **265-266**, 266-267
  - `ListPreference` **144-145**
  - `ListView` **106-107**
  - `Log` **148**, 170
  - `Long` **630**
  - `Math` 423
  - `MediaStore` **219-220**
  - `MediaStore.Images.Media` 219-220
  - `Menu` **143-144**, 161-162
  - `MenuInflater` **162-163**, 284-285
  - `MotionEvent` **186**, 207, **219**, 242
  - `MouseAdapter` 624-625
  - `MouseEvent` **617-618**
  - `MultiSelectListPreference` **144**
  - `NumberFormat` **73-74**, 84
  - `ObjectInputStream` **645-646**
  - `ObjectOutputStream` **645-646**
  - `Paint` **187-188**
  - `Path` **219-220**
  - `Preference` **144-145**
  - `PreferenceFragment` **143**, 175
  - `PreferenceManager` **144**, 160
  - `PrintHelper` 246
  - `R` **87**
  - `R.drawable` 87
  - `R.id` **87-88**
  - `R.layout` **87-88**
  - `R.string` **87-88**
  - `Random` 431-432, **432-433**
  - `Resources` **160**, 160, 168
  - `RuntimeException` **595**
  - `Scanner` **340**, **361**
  - `ScrollView` **269-270**
  - `SeekBar` 70-71, **72-73**, 85
  - `Sensor` **218**
  - `SensorEvent` **232-233**



SensorManager 230-231  
 SharedPreferences **106**, 119, 120  
 SharedPreferences.Editor **106**, 126  
 Short **630**  
 SimpleCursorAdapter **281-282**  
 SoundPool **186-187**, 195  
 SQLiteDatabase **265-266**  
 SQLiteOpenHelper **265-266**  
 StackTraceElement **603-604**  
 SurfaceHolder **187-188**, 195  
 SurfaceView **187-188**, 195  
 SwingWorker **654-655**  
 TableLayout **74-75**  
 TextView **39-40**, 52, 72-73, 85  
 Thread 186-187, 208  
 Throwable **594-595**, 603-604  
 Toast **146-147**, 164-165  
 TreeMap **641-642**  
 TreeSet **640-641**  
 UnsupportedOperationException **635**  
 Uri **119**, **128**  
 View **120**, 187-188  
 ViewGroup **269-270**  
 WindowManager **146-147**, 161-162  
 classes aninhadas 608  
 classes de evento 615-616  
 classes numéricas 630  
 classificação com um Comparador 639  
 classificação de aplicativos com estrelas 322-323  
 CLASSPATH, variável de ambiente 336  
 clear, método  
     de ArrayList<T> 488  
     de List<T> **635-636**  
 cliente de um objeto **366**  
 clique com o mouse 621  
 clique em um botão 609  
 clique nas setas de rolagem 623-624  
 clonando objetos  
     cópia profunda **546-547**  
     cópia rasa **546-547**  
 clone de Object, método 546-547  
 close, método da classe Cursor **284**  
 close, método da classe  
     SQLiteOpenHelper **299-300**  
 cobrança incorporada ao aplicativo 317  
     melhores práticas de segurança 317  
 código binário (bytecode) 335  
 código cliente 549-550  
 código de liberação de recurso 597  
 código de versão 311  
 código-fonte 2  
 código-fonte aberto 3  
 colchetes, [] **452**  
 coleção **487-488**, **629**  
     embaralhar 173

coleta de lixo 648-649  
 coleta de lixo automática 597-598  
 coletor de lixo **514**, 594, 597  
 Collection, interface **630**, 630, 633, 638  
     contains, método **633**  
     iterator, método **633**  
 Collections, classe 119, **148**, **631**  
     addAll, método 638  
     binarySearch, método 638  
     copy, método 638  
     fill, método 638  
     max, método 638  
     min, método 638  
     reverse, método 638  
     reverseOrder, método **639**  
     shuffle, método **148**, 638, **640**  
     sort, método **123**, **638**  
 College Loan Payoff Calculator, exercício do aplicativo 95-96  
 colocar em uma pilha **429-430**  
 colocar na pilha de retrocesso **276**  
 Color, classe 249  
     alpha, método **249**  
     argb, método **249-250**  
     blue, método **249**  
     green, método **249**  
     red, método **249**  
 color, pasta de um projeto Android **48**, **146**  
 colors.xml 151  
 Column, propriedade de LinearLayout 78  
 Column Count, propriedade de um GridLayout 76-77  
 coluna 476-477  
 colunas de um array bidimensional 476  
 comando Objective-C xix  
 comentário  
     fim de linha, // **333**, 334  
     linha única 334  
 comentário de fim de linha, // **333**, 334  
 comentário fim de linha 334  
 comentário tradicional **333**  
 Command Prompt 334  
 CommissionEmployee, classe derivada de Employee 561-562  
 commit, método da classe  
     FragmentTransaction **275**  
 Comparable<T>, interface 581, 638  
     compareTo, método 638  
 Comparator, interface **638**  
     compare, método **639**  
 Comparator, objeto 641-642  
 Comparator<String>, objeto de  
     String.CASE\_INSENSITIVE\_ORDER 123

compare, método da interface  
     Comparador **639**  
 compareTo, método de Comparable 638  
 compartilhamento de fotos 324-325  
 compartilhamento de vídeos 324-325  
 compilando aplicativos 309  
 compilando um aplicativo com várias classes 359  
 compilar 335  
 componente 16, 36  
 componente de tela xix  
 componente de um array **452**  
 componentes da interface gráfica do usuário não têm thread segura 146  
 componentes de interface gráfica do usuário  
     convenção de atribuição de nomes 75  
     criar por meio de programa 144  
     EditText **72-73**  
     ImageButton **106**, 113, **120**  
     ImageView **39-40**, 56  
     ScrollView **269-270**  
     SeekBar 70-71, **72-73**  
     TextView **39-40**, **49**, 52  
     ViewGroup **269-270**  
 componentes de software reutilizáveis 16, 36, 431-432  
 comportamento de uma classe 16, 34  
 composição **509-510**, 524  
 compra incorporada ao aplicativo 314  
 computação em nuvem 7  
 computadores na educação 450  
 Computer Assisted Instruction, exercício do aplicativo 181  
 comunicação em campo próximo (NFC) 8-9  
 concatenação **427**  
 concatenar strings 517  
 conclusão de E/S de disco 594-595  
 condição **346**  
 condição de continuação de loop **397**, 398-400, 402-404, 410  
 configuração de constantes como uma interface 570  
 configuração de dispositivo 13  
 configurar tratamento de evento 612-613  
 configurar um valor com *set* **366**  
 Configuration, classe **160-161**  
 confinamento de thread **653-654**  
 confundindo o operador de igualdade == com o operador de atribuição = 349  
 conhecimento de marca 316  
 conjunto de caracteres Unicode 397, 666  
 conjunto de colchetes externo 461

conjunto mais interno de colchete 461-462  
 console de jogo 5  
 constante 518-519  
   em uma interface 581  
   Math.PI 353  
 constante de caractere **409-410**  
 constante de enumeração **437-438**  
 constante de ponto flutuante 400  
 constante nomeada **457**  
 Constantes  
   MODE\_PRIVATE **122**  
   MODE\_WORLD\_READABLE **122**  
   MODE\_WORLD\_WRITEABLE **122**  
 construtor **358**, 367  
   chamar outro construtor da mesma classe usando this 505  
   lista de parâmetros 368  
   sem argumentos **505-506**  
   sobrecarregado **501-502**  
   vários parâmetros 369  
 construtor padrão **367**, 508, 529  
 construtor sem argumentos **505**, 507  
 construtores não podem especificar um tipo de retorno 368  
 construtores sobrecarregados **501**  
 consumir um evento **613-614**  
 conta para cobrança **319**  
 conta poupança 400-401  
 contador **383**  
 contagem baseada em zero **455**  
 contains, método da classe  
   ArrayList<T> 488, **490**  
 contains, método de Collection **633**  
 containsKey, método de Map **643-644**  
 Content Description, propriedade **59-60**, 114  
 ContentResolver, classe **219-220**  
 ContentValues, classe **299-300**  
 conter outros Views **269-270**  
 Context, classe **119**  
   getSharedPreferences, método **122**  
   startActivity, método **107**, 128  
 controlar instalações de aplicativo 322  
 controle 15, 36  
 controle de programa **379**  
 controle de versão de seu aplicativo 309  
 controle deslizante 72-73  
 controles 608  
 conversão de atribuição de nomes  
   componentes de interface gráfica do usuário 75-76  
 convenções alternativas de atribuição de nomes de recurso 61-62  
 conversão  
   downcast **550-551**  
   operador **390-391**, 430-431

conversão (double) **390-391**  
 conversão de encaixotamento (boxing) **630**  
 conversão de tipo **390-391**  
 conversão explícito **390-391**  
 conversão implícita **390-391**  
 conversão unboxing **630**  
 Cooking with Healthier Ingredients, exercício do aplicativo 305-306  
 cópia profunda **546-547**  
 cópia rasa **546-547**, 547-548  
 copiando objetos  
   cópia profunda **546-547**  
   cópia rasa **546-547**  
 copy, método de Collections 638  
 cor 187-188  
 corners, elemento de uma forma **268-269**  
 corpo  
   de um loop 383  
   de um método **334**  
   de uma declaração de classe **333**  
   de uma instrução if 346  
 correspondendo ao bloco catch 592  
 cos, método de Math 424  
 cosseno 424  
 cosseno trigonométrico 424  
 Country Quiz, exercício do aplicativo 180  
 craps (jogo de cassino) 181, 434, 450  
 Craps Game, exercício do aplicativo 181  
   modificação 181  
 crash reports 322-323  
 Create New Android String, caixa de diálogo 53  
 createBitmap, método da classe Bitmap **238-239**  
 createChooser, método da classe Intent **131**  
 createFromStream, método da classe Drawable **173**  
 criando e inicializando um array 455  
 criando um banco de dados 298-299  
 criar componentes de interface gráfica do usuário por meio de programa 144-145  
 criar um objeto de uma classe 358  
 criar um pacote 518-519  
 crivo de Eratóstenes 493  
 Crossword Puzzle Generator, exercício do aplicativo 306-307  
 <Ctrl>-d 406-407  
 <Ctrl>-z 406-407  
 Cursor, classe **265**, 296, 301  
   close, método **284-285**  
   getColumnIndex, método **296-297**  
   getColumnIndexOrThrow, método **296-297**

getString, método **296-297**  
 moveToFirst, método **296-297**  
 cursor 334, 336  
 cursor de saída 336  
 cursor de tela 337  
 CursorAdapter, classe **265**, 281  
   changeCursor, método 284, **284**  
   getCursor, método **284-285**  
 CursorFactory, classe 302-303  
 curva Bezier 243-244  
 curva Bezier quadrática 243-244  
  
**D**  
 dados persistentes **644-645**  
 Dalvik Debug Monitor Service (DDMS) 312-313  
 data 431-432  
 Date, exercício da classe 522  
 Daydream 10-11  
 DDMS (Dalvik Debug Monitor Server) 312-313  
 decisão **346**  
 declaração  
   classe **333**  
   import **339**, 340  
   método **334**  
 declaração de método 426  
 declarar um método de uma classe **356**  
 decremento de uma variável de controle **397-398**  
 default, case em um switch **407**, 409  
 deficiências 39-40, 59  
 Deitel® Buzz Online Newsletter ([www.deitel.com/newsletter/subscribe.html](http://www.deitel.com/newsletter/subscribe.html)) xxv, 328  
 delete, método da classe SQLiteDatabase **301-302**  
 depurando exceções de login **148**, 170  
 desenhar  
   círculos 187-188  
   linhas 187-188  
   texto 187-188  
 desenrolar a pilha de chamada de métodos 601-602  
 desenvolvimento de aplicativos xix  
 deslocar (números aleatórios) **432**  
 despachar um evento **617-618**  
 detalhes específicos 549-550  
 detecção de colisão 187, 197-200  
 detecção de colisão simples 199-200  
 detecção de rosto 9-10  
 Dev Guide 309  
 Device Screen Capture, janela 313  
 diagrama de classes em UML da hierarquia de interfaces Payable 572  
 DialogFragment, classe **143-144**, 175  
   onCreateDialog, método **175**  
   show, método **175**



DialogInterface, classe **119**  
 DialogInterface.OnClickListener, interface **119**  
 diâmetro 353  
 dica em um EditText 271-272  
 diferenciação de maiúsculas e minúsculas **333**  
 Digital Clock, exercício do aplicativo 212  
 digitando em um campo de texto 609  
 dígito 340  
 Digits, propriedade de um EditText 80  
 dimens.xml 112  
 diretório 645-647  
     nome 645-646  
 diretório pai 646-647  
 diretório-raiz **645-646**  
 Display, classe **146-147**, 161-162  
 dispositivo de tela grande 8-9  
 distância entre valores (números aleatórios) 433-434  
 distribuição 462-463  
 dividir por zero 588  
 divisão 343, 344  
 divisão inteira **343**  
 do...while, instrução de repetição 380, **402-403**  
 documentação  
     *Android Design* 32  
     *App Components* 32  
     *Class Index* 32  
     *Data Backup* 32-33  
     *Debugging* 32-33  
     *Get Started with Publishing* 32-33  
     *Getting Started with Android Studio* 32-33  
     *Google Play Developer Distribution Agreement* 32-33  
     *Launch Checklist (for Google Play)* 32-33  
     *Managing Projects from Eclipse with ADT* 32-33  
     *Managing Your App's Memory* 32-33  
     *Package Index* 32  
     recursos de aplicativo **52**  
     *Security Tips* 32-33  
     *Tools Help* 32-33  
     *Using the Android Emulator* 32  
 documentação da API Java 342  
     download 342  
 documentação da API Java SE 6 431-432  
 documentação de API 431-432  
 documentação do Eclipse (www.eclipse.org/documentation) xiv-xv  
 documentação para desenvolvedor  
     *Keeping Your App Responsive* 32-33  
     *Launch Checklist* 310

*Performance Tips* 32-33  
*Signing Your Applications* 312-313  
*Tablet App Quality Checklist* 310  
 documentação para desenvolvedor Java (www.oracle.com/technetwork/java/javase/downloads/index.html) xiv-xv  
 documentar um programa **332**  
 doInBackground, método da classe AsyncTask 282, **283**, 284, 295  
 Doodlez, aplicativo ix, **19**  
 Doodlez, exercício do aplicativo 256  
 Double, classe **630**  
 double, tipo primitivo **340**, **369**, **370**, 387-388, 666  
     promoções 430-431  
 downcast 566-567  
 dp (pixels independentes de densidade) 54-55, **54-55**, 54-55  
 Drawable, classe **173**  
     createFromStream, método **173**  
 drawable, pasta de um projeto Android **48**  
 drawBitmap, método da classe Canvas **239-240**  
 drawCircle, método da classe Canvas **203-204**  
 drawLine, método da classe Canvas **204-205**  
 drawPath, método da classe Canvas **239-240**, 244  
 drawRect, método da classe Canvas **203-204**  
 drawText, método da classe Canvas **203-204**  
 duplo igual, == 349

## E

e, método da classe Log **170**  
 E condicional, && **410**, 412  
     tabela-verdade 411-412  
 E lógico booleano, & **410**, **412**  
 Eclipse  
     importar projeto 70, 100, 140, 185, 263  
     janela Outline 70-73  
     vídeo de demonstração 332  
 edit, método da classe SharedPreferences **126**  
 EditText  
     Digits, propriedade 80-81  
     Ems, propriedade 80-81  
     Max Length, propriedade 80-81  
 EditText, classe **72-73**, 85  
     Hint, propriedade **112**, 114  
     IME Options, propriedade **112**, 114  
     restringir o número máximo de dígitos 72-73  
     tipo de entrada 77-78

EditText, restringir o número máximo de dígitos em um 72-73  
 EditText, tipo de entrada de um 77  
 efeito colateral **412-413**  
 efeitos sonoros 186-187  
 efetuar operações de forma concorrente **647-648**  
 efetuar um cálculo 350  
 elemento de preenchimento de uma forma **268-269**  
 elemento de tabela 476-477  
 elemento de um array **452**  
 elemento zero **452**  
 elementos gráficos x-xi, 13  
 Eliminação de duplicatas 493  
 eliminar vazamentos de recurso 597  
 em linha reta **343**  
 embaralhamento de cartas  
     Fisher-Yates 465-466  
 embaralhamento Fisher-Yates 465  
 embaralhar 462-463  
     algoritmo 640  
 embaralhar uma coleção 173  
 empacotamento de objetos fluxo **647**  
 Employee, classe que implementa Payable 576-577  
 Employee, classes, programa de teste da hierarquia de 564  
 Employee, superclasse abstrata 557  
 empresas de projeto de ícone 99designs 311  
     Aha-Soft 311  
     Androidicons 311  
     Elance 311  
     glyphlab 311  
     Iconiza 311  
 Ems, propriedade de um EditText 80  
 emulador 13-14, 310  
     gestos 15  
 encapsulamento **18**  
 engenharia reversa 311  
 Enter (ou Return), tecla 616-617  
 entrada de dados a partir do teclado 350  
 entrada numérica 72-73  
 enum **437-438**  
     classe EnumSet 514  
     constante 512  
     construtor **512**  
     declaração **512**  
     método values **513**  
     palavra-chave 437-438  
 enumeração **437-438**  
 EnumSet, classe **514**  
     método range **514**  
 enviar mensagem 367  
 enviar uma mensagem para um objeto 16-17, 34-35

- equals, método
    - Arrays, da classe **485-486**
    - Object, da classe 546-547
  - eraseColor, método da classe Bitmap **253-254**
  - erro de compilação **333**
  - erro de lógica 341, **383**, 399-400
  - erro de lógica fatal **383**
  - erro de lógica não fatal **383**
  - erro de lógica no tempo de execução (em runtime) 341
  - erro de sintaxe **333**, 335
  - erro do compilador **333**
  - erro fatal **383**
  - erro por um **399-400**
  - erro síncrono **594-595**
  - Error, classe **595**
  - escala de temperatura Kelvin 626-627
  - escalar **468-469**
  - escopo 399-400
  - escopo de uma declaração **438-439**
  - escopo de uma variável **399-400**
  - escopo de variável 399-400
  - esfera 444-445
  - espaço em branco **333**, 334, 349
  - especialização **524**
  - especificações para captura de tela 312-313
  - especificador de formato
    - numeração em um recurso de String 149
    - vários em um recurso de String 149
  - especificadores de formato **338**
    - %2f para números de ponto flutuante com precisão 391-392
    - %d **341**
    - %f 353, **371**
    - %s **338**
  - espessura de linha 187-188
  - estado coerente 501-502
  - estado salvo 87
  - estender uma classe **524**
  - estimular as vendas 316
  - estouro 594-595
  - estouro aritmético 594-595
  - estratégia dividir para conquistar 423
  - estrutura aninhada de um layout 77
  - estrutura de dados **452**
  - estrutura de sequência **379**
  - estruturas de dados predefinidas 629
  - EventListener, classe **617-618**
  - evento 581, **609**
  - evento assíncrono **594-595**
  - evento de arrastar 243-244
  - evento de mouse **617-618**
  - evento de pressionamento longo 219
  - evento de tecla **617-618**
  - evento de toque 219-220, **241-242**
  - evento de toque com movimento rápido (fling) 219-220
  - evento de toque para rolar 219-220
  - eventos **5**
  - eventos de toque
    - movimento rápido (fling) 219-220
    - pressionamento longo 219-220
    - rolagem 219-220
    - simples 186-187
  - eventos de toque simples 186-187
  - exame de código 2
  - exceção **461-462**, 587
    - parâmetro 462-463
    - rotina de tratamento **461-462**
    - tratando 460
  - exceção não capturada **592**
  - exceção verificada **595**
  - Exceções 461-462
    - IndexOutOfRangeException 461
  - exceções não verificadas **595**
  - Exception, classe **594-595**
  - exclusão mútua **652-653**
  - execSQL, método da classe SQLiteDatabase **303-304**
  - execução sequencial **379**
  - executar uma ação 334
  - execute, método da classe AsyncTask **282-283**
  - execute, método da interface Executor **659-650**, 652-653
  - ExecutionException, classe 657
  - Executor, interface **659-650**
    - execute, método **659**, 652
  - Executors, classe **659-650**
    - newCachedThreadPool, método **650**
  - ExecutorService, interface **659-650**
    - shutdown, método **652-653**
  - Exemplos xxv
  - exercício de Computer Assisted Instructions (CAI) 181
  - exercício de falha de construtor 606
  - exercício do aplicativo **Recipe** 305
  - exercício hierarquia de formas 584
  - exibir saída 350
  - exibir uma linha de texto 334
  - exists de File, método 646-647
  - exit, método da classe System **597**
  - exp, método de Math 424
  - Expense Tracker, exercício do aplicativo 305-306
  - Explore by Touch **39-40**, **59**
  - expressão **341**
  - expressão condicional **381**
  - expressão de acesso ao array **452**
  - expressão de controle de um switch **407**
  - expressão de criação de array **453**
  - expressão de criação de instância de classe **358**, 368
  - expressão inteira 409-410
  - expressão inteira constante **403**, 409
  - extends, palavra-chave **527**, 537-538
  - extensão de nome de arquivo .java 356
  - extensibilidade 549-550
  - extrair a pilha de retrocesso 276
  - extrair de uma pilha **429-430**
  - extras de intent 131
- F**
- fabricante de equipamento original (OEM) 4
  - fabricante de equipamento original OEM 4
  - Facebook 104-105, **324-325**
    - página da Deitel 324-325
  - Fahrenheit 626-627
    - equivalente de uma temperatura em Celsius 545
  - false, palavra-chave **346**, **381**
  - false, retornando de uma rotina de tratamento de evento **240-241**
  - fator de escala (números aleatórios) 432-434
  - fatura da operadora 315
  - Favorite Twitter Searches, exercício do aplicativo melhorado 305-306
  - Favorite Websites, exercício do aplicativo 135
  - fazendo downcast **550-551**
  - fazendo loop **386**
  - fazer referência a um objeto **366**
  - fazer seu ponto (jogo de craps) 434
  - fazer uma animação manualmente 186
  - fechar uma janela 609
  - ferramentas
    - logcat **148**
  - fila 630, 640-641
  - fila de espera 630, 640-641
  - File, classe **645-646**
    - canRead, método 646-647
    - canWrite, método 646-647
    - exists, método 646-647
    - File, métodos 646-647
    - getAbsolutePath, método 646-647
    - getName, método 646-647
    - getParent, método 646-647
    - getPath, método 646-647
    - isAbsolute, método 646-647
    - isDirectory, método 646-647
    - lastModified, método 646-647
    - length, método 646-647
    - list, método 646-647
  - File, métodos 646-647
  - FileInputStream, classe **645**, 647

FileOutputStream, classe **645**, 647  
 FileReader, classe **645-646**  
 FileWriter, classe **645-646**  
 fill, método  
   da classe Arrays **485**, 487  
   da classe Collections **638**  
 filtro de intenção 107-108  
 “fim da entrada de dados” 387-388  
 fim de arquivo (EOF)  
   indicador **406-407**  
   marcador **644-645**  
 final  
   classe **569**  
   classes e métodos 569  
   método **569**  
   palavra-chave 409, **424**, 457, 518, 569  
   variável 457  
   variável local 623-624  
 final, classe não pode estender uma classe 569  
 final, variável local para uso em uma classe interna anônima 129  
 finalize, método 546-547  
 finally  
   bloco **592**, 597-598  
   cláusula **597-598**  
   palavra-chave **592**  
 findFragmentById, método da classe Activity **144-145**, 161-162  
 Fireworks Designer, exercício do aplicativo 212  
 flag de formatação sinal de subtração - **401**  
 flag de formatação vírgula (,) **402**  
 Flag Quiz Game, aplicativo ix  
   exercício 180  
 Flickr Searches, exercício do aplicativo 135  
   melhorado 135  
 float  
   promoções de tipo primitivo 430-431  
   tipo primitivo **340**, **369**, **370**, 666  
 Float, classe **630**  
 floor de Math, método 424  
 FlowLayout, classe **611**  
 fluxo **599-600**  
   fluxo baseado em byte **644-645**  
   fluxo baseado em caractere **644-645**  
   fluxo de áudio de música 186-187, 195  
   fluxo de bytes **644-645**  
   fluxo de controle 390-391  
   fluxo de entrada padrão (System.in) **340**  
   fluxo de erro padrão **592**, **599-600**  
   fluxo de saída padrão (System.out) **334**

fluxo de saída padrão **599-600**  
 fluxo para reproduzir música 195  
 fluxos de áudio 186-187  
   música 186-187  
 foco **609**  
 fonte de tamanho médio 77-78  
 for, instrução aninhada 459, 478, 479, 483  
 for, instrução melhorada **466-467**  
 format, método da classe  
   NumberFormat **87-88**  
 format, método da classe String **498**  
 formatação de inteiro decimal 341  
 formato de hora padrão 498  
 formato de hora universal 496-498  
 formato de relógio de 24 horas 496  
 formato tabular 456  
 Formatter, classe **645-646**  
 Fortune Teller, exercício do aplicativo 257-258  
 fóruns 32-33  
   Android Forums 33-34  
   Stack Overflow 32-33  
 Fractal, exercício do aplicativo 257  
 Fragment, ciclo de vida de 218, 280-285, 287-288, 292, 294  
 Fragment, classe 71-72, **143-144**  
   getActivity, método 168-169  
   getResources, método **168-169**  
   onActivityCreated, método 190  
   onAttach, método **218**, 249, 280, 287, 292  
   onCreate, método **144-145**, 175-176  
   onCreateOptionsMenu, método **233**  
   onCreateView, método **144**, 166, 190  
   onDestroy, método **186**, 191  
   onDetach, método **218**, 249, 280, 287, 292  
   onOptionsItemSelected, método **233-234**  
   onPause, método de ciclo de vida 231-232  
   onPause, método **186**, 191  
   onResume, método 282-283, 294  
   onSaveInstanceState, método **264-265**, 294  
   onStart, método de ciclo de vida **230-231**  
   onStop, método **284-285**  
   onViewCreated, método **280-281**  
   setArguments, método **276**  
   setRetainInstance, método **281**  
 Fragment, layout 152-153  
 Fragment, métodos de ciclo de vida de 249  
 FragmentManager, classe **144-145**  
   beginTransaction, método **275**

getFragmentByTag, método **175**  
 popBackStack, método **276**  
 fragmento 8-9, **143-144**  
 FragmentTransaction, classe **144**, **264**, 275, 276  
   add, método **275**  
   addToBackStack, método **276**  
   commit, método **275**  
   replace, método **276**  
 FrameLayout, classe **188-189**  
 framework de coleções **629**  
 Froyo (Android 2.2) 7  
 Fullscreen Activity, template **44**  
 função **423**  
 funcionalidade do emulador 15

## G

Game of Snake, exercício do aplicativo 257-258  
 gen, pasta de um projeto Android **87**  
 generalidades 549-550  
 gerenciador de layout 611  
   FlowLayout 611  
 gesto 5  
   arrastamento 5  
   movimento rápido 5  
   pressionamento longo 5  
   toque 5  
   toque duplo 5  
   toque duplo rápido 5  
   toque rápido 5  
   zoom de pinça 5  
 gestos e controles do emulador 15  
 GestureDetector.  
   OnDoubleTapListener, interface **219**, 240  
   GestureDetector.OnGestureListener, interface **219-220**  
 GestureDetector.  
   SimpleGestureListener, classe **219**, 240  
   onSingleTap, método **240-241**  
 get, método  
   da classe ArrayList<T> **490**  
   da interface List<T> **633**  
   da interface Map **643-644**  
 get, método 366, 501-502  
 getAbsolutePath, método da classe File 646-647  
 getActionCommand, método da classe ActionEvent **614**, 621  
 getActionIndex, método da classe MotionEvent **242-243**  
 getActionMasked, método da classe MotionEvent **242-243**  
 getActivity, método da classe Fragment 168-169  
 getAll, método da classe SharedPreferences **122**

- getAssets, método da classe ContextWrapper **169-170**, 173
  - getAssets, método da classe ContextWrapper **169-170**, 173
  - getClass, método de Object 547-548, **568**
  - getClassName, método da classe StackTraceElement **603-604**
  - getColumnIndex, método da classe Cursor **296-297**
  - getColumnIndexOrThrow, método da classe Cursor **296-297**
  - getConfiguration, método da classe Resources **160-161**
  - getCursor, método da classe CursorAdapter **284-285**
  - getDefaultSensor, método da classe SensorManager **230-231**
  - getFileName, método da classe StackTraceElement **603-604**
  - getFragmentByTag, método da classe FragmentManager **175**
  - getFragmentManager, método da classe Activity **144-145**, 161-162, 175
  - getHolder, método da classe SurfaceView **195**
  - getItemID, método da classe MenuItem **234-235**
  - getItemID, método da classe MenuItem **234-235**
  - getLineNumber, método da classe StackTraceElement **603-604**
  - getListView, método da classe ListFragment **281-282**
  - getListViewDefault, método Para Font> da classe ListActivity **123**
  - getMenuInflater, método da classe Activity **162-163**
  - getMessage, método da classe Throwable **603-604**
  - getMethodName, método da classe StackTraceElement **603-604**
  - getName, método da classe Class 547-548, **568**
  - getName, método da classe File 646
  - getParent, método da classe File 646
  - getPassword, método da classe JPasswordField **614-615**
  - getPath, método da classe File 646
  - getPointerCount, método da classe MotionEvent **243-244**
  - getResources, método da classe Activity **160-161**
  - getResources, método da classe Fragment **168-169**
  - getSelectedIndex, método da classe JComboBox **624-625**
  - getSharedPreferences, método da classe Context **122**
  - getSource, método da classe EventObject 614-615
  - getSource, método da classe EventObject 614-615
  - getStackTrace, método da classe Throwable **603-604**
  - getStateChange, método da classe ItemEvent **624-625**
  - getStateChange, método da classe ItemEvent **624-625**
  - getString, método da classe Activity **127**, 130
  - getString, método da classe Cursor **296-297**
  - getString, método da classe Resources **168-169**
  - getString, método da classe SharedPreferences **127**
  - getStringSet, método da classe SharedPreferences **164-165**
  - getSystemService, método da classe Activity 230-231
  - getSystemUiVisibility, método da classe View **240-241**
  - getWritableDatabase, método da classe SQLiteOpenHelper **298-299**
  - getX, método da classe MotionEvent 243-244
  - getY, método da classe MotionEvent 243-244
  - Google Cloud Messaging 7
  - Google Maps 6
  - Google Play
    - idioma 320
    - locais 321-322
    - preço 321-322
  - Google Play **11-12**, 309, 310, 315, 318, 325-326
    - capturas de tela 320
    - conta de publicador 317-318
    - crash report 322-323
    - elemento gráfico promocional 320
    - ícone de aplicativo de alta resolução 320
    - países 321-322
    - publicar 319, 320
    - Publish an Android App on Google Play 320
    - taxas 319
    - vídeo promocional 313-314, 320
  - Google Play Developer Console 322-323
  - Google Play Developer Program Policies* 318
  - Google Waller 309, 315, **319**
  - conta 321-322
  - Google+ 104-105
  - goto, eliminação de 379
  - goto, instrução **379**
  - GPS ix-x
  - gráfico de barras 458, 459
  - Graphical Layout, editor **39-40**, 46-50
  - Graphical Layout, editor no Android Developer Tools **39**, 46, 48, 49
  - gravação de áudio ix-x
  - gravável 646-647
  - Gravity (layout), propriedade 80-81
  - Gravity, propriedade de um componente **55**
  - green, método da classe Color **249**
  - GridLayout
    - Column Count, propriedade 76-77
    - Orientation, propriedade 76-77
    - Use Default Margins, propriedade 76
  - GridLayout, classe **72-73**, **110**
  - documentação 74-75
  - grupos de discussão 32-33
    - Android Developers 32-33
  - grupos de discussão do Open Source Project 3
  - GUI (interface gráfica do usuário) 581
  - componente **608**
- ## H
- habilitar/desabilitar VoiceOver 59-60
  - Handler, classe **146-147**
    - postDelayed, método **146**, 175
  - Hangman Game, exercício do aplicativo 257-258
  - hashCode, método de Object 547-548
  - HashMap, classe **641-642**
    - keySet, método **643-644**
  - HashSet, classe **640-641**
  - Hashtable, classe **641-642**
  - hashtag 324-325
  - hasNext, método
    - Iterator, da interface **633**, 635
    - Scanner, da classe **407-408**
  - hasPrevious, método de ListIterator **635-636**
  - herança **18**, **524**
    - exemplos 525
    - hierarquia **525**, 553-554
    - hierarquia para MembroDaComunidade 525
    - palavra-chave extends **527**, 537
  - hierarquia de classes **524**, 553-554
  - hierarquia de coleções 630
  - Hint, propriedade de um EditText **112**, 114
  - Holo, interface de usuário 8-10
  - Holo Dark, tema 42-43
  - Holo Light, tema 42-43

Holo Light With Dark Action Bars, tema 42-43  
 Home, botão 22-24  
 Horse Race with Cannon Game, exercício do aplicativo 212  
 HourlyEmployee, classe derivada de Employee 559-560  
 HugeInteger, classe, exercício 522

## I

ícone 309, **310**  
 ícone de transição **621**  
 Id, propriedade de um layout ou componente **50-51**  
 IDE (ambiente de desenvolvimento integrado) 13-14, 36  
 IDE Android Developer Tools 38-40  
 IDE Eclipse **2**  
 identificação de evento **618-619**  
 identificador **333**, 340  
 Identificador válido 340  
 IEEE 754 ([grouper.ieee.org/groups/754/](http://grouper.ieee.org/groups/754/)) 666  
 if, instrução de seleção simples **346**, 380, 403-404  
 if...else, instrução de seleção aninhada **381**  
 if...else, instrução de seleção dupla **380**, 389, 403  
 ignorando o elemento zero do array 461  
 IllegalArgumentException, classe **497**  
 ImageButton, classe **106**, 113, 120  
 imagens x-xi  
 ImageView, classe **39-40**, 56  
   Adjust View Bounds, propriedade 154  
   Scale Type, propriedade 154  
 IME, propriedade Options de um EditText **112**, 114  
 IME Options 271-272  
 impedir a exibição do teclado virtual na inicialização do aplicativo 133  
 impedir a exibição do teclado virtual quando o aplicativo é carregado 109  
 implementação de uma função 557  
 implementar uma interface **548**, **570**, 576  
 implements, palavra-chave **570**, 575  
 Import, caixa de diálogo 21, 70-71, 185, 263-264  
 import, declaração **339**, 340, 362  
 importar um projeto já existente para o Eclipse 70-71, 100, 140, 185, 263-264  
 imprimir em várias linhas 336  
 imprimir uma linha de texto **334**  
 incremento  
   de uma variável de controle **397**  
   expressão 410-411  
   operador, ++ 395-396  
   uma variável de controle 398-399  
 indexOf, método da classe ArrayList<T> 488  
 IndexOutOfBoundsException, classe 462  
 índice (subscrito) **452**  
 índice 460  
 índice de array fora do limite 594  
 índice zero **452**  
 i-Newswire 326  
 inflando uma interface gráfica do usuário **87-88**  
 inflar a interface gráfica do usuário 195-196  
 inflate, método da classe LayoutInflater **166-167**  
 inflate, método da classe MenuInflater **162-163**  
 informação de caminho **645-646**  
 informação em nível de classe **514**  
 informações em gráficos 459  
 informando sobre erros 3  
 inicialização no início de cada repetição 392-393  
 inicializador de array **456**  
   aninhado **477**  
   para array multidimensional 477  
 inicializando arrays bidimensionais em declarações 478  
 inicializar uma variável em uma declaração **340**  
 Input Type 271-272  
 InputMethodManager, classe **120**  
 InputMismatchException, classe **589**, 591  
 InputStream, classe 173, 647-648  
   setImageDrawable, método **173**  
 insert, método da classe SQLiteDatabase **299-300**  
 insertImage, método da classe MediaStore.Images.Media, método **219**  
 instanceof, operador **566-567**  
 instância **16-17**  
 instância de uma classe 363  
 instanciando um objeto de uma classe **356**  
 instrução **334**, 357  
 instrução assistida por computador (CAI): reduzindo a fadiga dos alunos 450  
 instrução assistida por computador (CAI): variando os tipos de problemas 450  
 instrução assistida por computador (CAI) 450

instrução continue **410**, 410  
 instrução de atribuição 341  
 instrução de controle **379**  
 instrução de declaração de variável **340**  
 instrução de repetição **379**, **380**  
   do...while 380, 402-403  
   for 380  
   while 380, **383**, 386, 389, 397  
 instrução de repetição for 380, **398**, 401  
   aninhada 459  
   cabecinho **399-400**  
   melhorada **466-467**  
 instrução de seleção **379**, **380**  
   if 380, 403-404  
   if...else **380**, 389-390, 403-404  
   switch 380, **403-404**  
 instrução de seleção dupla **380**  
 instrução de seleção múltipla **380**  
 instrução de seleção múltipla switch 380, **403-404**  
   case default **407-408**, 409-410  
   comparando Strings 409-410  
   expressão de controle **407-408**  
   rótulo case 407-408  
 instrução de seleção simples **380**  
 instrução vazia (um ponto e vírgula, ;) **349**  
 instruções  
   aninhadas **391-392**  
   break **407-408**, 410-411  
   continue **410-411**  
   de loop **380**  
   do...while 380, **402-403**  
   for 380, **398-399**, 401-402  
   for melhorada **466-467**  
   if...else **380**, 389-390, 403-404  
   if...else aninhado **381**  
   if **346**, 380, 403-404  
   instrução de controle **379**  
   repetição **379**, **380**  
   seleção **379**, **380**  
   seleção dupla **380**  
   seleção múltipla **380**  
   seleção simples **380**  
   switch 380, **403-404**  
   try **461-462**  
   vazias **349**  
   while 380, **383**, 386, 389, 397  
 instruções de controle aninhadas **391**  
   problema dos resultados de exame 393  
 int, tipo primitivo **340**, 387, 395, 403, 666  
   promoções 430-431  
 Integer, classe **630**  
 integerPower, método 447-448



inteiro **338**  
 array 456  
 divisão **387**  
 quociente 343  
 valor 340  
 Intent, classe **107-108**, 119  
   ACTION\_SEND, constante **130**  
   ACTION\_VIEW, constante **128**  
   Bundle 131  
   createChooser, método **131**  
   explícita **107-108**, **148**  
   implícita **107-108**  
   método putExtra **131**  
 Intent explícita **107**, **148**, 162  
 Intent implícito **107-108**  
 interface, palavra-chave **570**  
 interface 548-549, 571-572, 579  
   declaração **570**  
   implementando métodos em Java **90**  
 interface de marcação (tagging)  
   571-572  
 interface de programação de  
   aplicativo (API) **422**  
 interface Editable 85  
 interface gráfica do usuário (GUI)  
   581, **608**  
 interface gráfica do usuário com  
   thread segura 146-147  
 interface gráfica do usuário portátil  
   431-432  
 interface Iterator **631**  
   método hasNext **633**  
   método next **633**  
   método remove **633**  
 interfaces **570**  
   ActionListener **613-614**, 617-618  
   AdapterView.OnItemClickListener  
     **120**, 281-282  
   AdapterView.OnItemLong-  
     ClickListener **120**  
   Collection **630**, 630, 638  
   Comparable 581, **638**  
   Comparator **638**  
   DialogInterface.OnClickListener  
     **119**  
   Editable 85  
   Executor **659-650**  
   ExecutorService **659-650**  
   GestureDetector.  
     OnDouble-TapListener **219**, 240  
   GestureDetector.  
     OnGestureListener **219-220**  
   Iterator **631**  
   KeyListener **618-619**  
   List **148**, 630, **635-636**  
   ListIterator **631**  
   Map 630, 641-642  
   MouseListener **617-618**

MouseMotionListener **618**, 624  
 ObjectInput **647-648**  
 ObjectOutput **647-648**  
 OnSeekBarChangeListener 89  
 Queue **630**, **640-641**  
 Runnable 146-147, **648-649**, 581  
 SeekBar.OnSeekBarChangeListener  
   73-74, 85, 249-250  
 SensorEventListener 231-232  
 Serializable 581  
 Set **148**, **630**, **640-641**  
 SortedMap **641-642**  
 SortedSet **641-642**  
 SurfaceHolder.Callback **187-188**,  
   195, 206  
 SwingConstants 581  
 TextWatcher **73-74**, 85  
 View.OnClickListener **120**  
 WindowListener 624-625  
 internacionalização ix, **39**, **61**, 73  
 interpretador java 335  
 interrupt, método da classe Thread  
   **650-651**  
 InterruptedException, classe **650-651**  
 invalidate, método da classe View  
   **238-239**  
 isAbsolute, método de File 646-647  
 isDirectory, método de File 646  
 isEmpty, método da classe Map **644**  
 item 320  
 iteração (loop) de um loop for 461  
 iteração **386**  
   de um loop **397-398**, 410-411  
 iterador bidirecional **635**  
 iterator, método de Collection **633**

## J

J2ObjC 323-324  
 janela de comando **334**  
 janela de terminal **334**  
 Java, código xix  
 Java, comando **332**  
 Java, site de 431-432  
 java.awt.event, pacote 431, **615**, 624  
 java.io, pacote 173, 431, **645**  
 java.lang, pacote **340**, 423, 431,  
   527, 546, 648  
   importado em cada programa Java  
     340  
 java.math, pacote 370  
 java.text, pacote **73-74**, 84  
 java.util, pacote 148, **339**, 431, 488  
 java.util.concurrent, pacote 659,  
   653  
 Java 5  
 Java Abstract Window Toolkit Event,  
   pacote **431-432**  
 Java API Interfaces 580

Java Concurrency Package **431-432**  
 Java Development Kit (JDK) 334  
 Java Input/Output Package **431-432**  
 Java Language Package **431-432**  
 Java Resource Centers 334  
 Java SE 7 409-410  
   Strings em instruções switch 409  
 Java Swing GUI Components  
   Package **431-432**  
 Java Utilities Package **431-432**  
 Java Virtual Machine (JVM) 332  
 javac, compilador 335  
 Java™ Language Specification 344  
 javax.swing, pacote 431, 617  
 javax.swing.event, pacote **615**, 624  
 JButton, classe **618-619**, 621  
 JComboBox, classe **622**  
   getSelectedIndex, método **624**  
   setMaximumRowCount, método **623**  
 JComboBox, índice de um **623-624**  
 JComboBox que exibe uma lista de  
   nomes de imagem 622  
 JComponent, classe 617-618, 622  
 JDK 334  
 jogando 431-432  
 jogo “adivinha o número” 545, 626  
 jogo de dados 434-435  
 jogos 31  
 jogos de cartas 462-463  
 JPasswordField, classe **609**, 614-615  
   getPassword, método **614-615**  
 JTextComponent, classe **609**, 612-613  
   setEditable, método **612-613**  
 JTextField, classe **609**, 613, 616  
   addActionListener, método **613**  
 JTextFields e JPasswordField 609  
 juros compostos 400-401

## K

Kaleidoscope, exercício do aplicativo  
 257-258  
 KeyEvent, classe **618-619**  
 KeyListener, interface **618-619**  
 keySet, método da classe HashMap  
   **643-644**  
 keySet, método da interface Map **122**

## L

Label For, propriedade de um  
 TextView 80-81  
 Labyrinth Game, aplicativo: exercício  
   de código-fonte aberto 257-258  
 lançamento de dois dados 436-437  
 lançamento de moeda 432-433, 545  
 lançar uma exceção **461-462**, **497**,  
   587, 591  
 language package 431-432  
 largura de campo **401-402**

largura de uma coluna 74-75  
 lastModified, método da classe File 646-647  
 layout, pasta de um projeto Android 48  
 layout 13  
 LayoutInflater, classe 144-145  
   método inflate 166-167  
 layouts  
   GridLayout 72-73  
   LinearLayout 72-73  
 layouts ix  
   activity\_main.xml 49-50  
   GridLayout 110  
   RelativeLayout 48  
   TableLayout 74-75  
 legibilidade 332  
 length, campo de um array 453  
 length, método de File 646-647  
 length, variável de instância de um array 453  
 letra maiúscula 333, 340  
 letra minúscula 333  
 letras maiúsculas e minúsculas como  
   no título de um livro 618-619  
 liberar recursos 296-297  
 liberar um recurso 597-598  
 licença de código ix-x  
 licença para Android 4  
 LIFO (último a entrar, primeiro a sair  
   - last-in, first-out) 429-430  
 limite de crédito em uma conta 418  
 limpeza ao terminar 546-547  
 LinearLayout, classe 72-73  
 linguagem orientada a objetos 18  
 linguagens fortemente tipadas 396  
 linha de comando 334  
 linha em branco 333  
 linhas de um array bidimensional  
   476-477  
 LinkedList, classe 631  
   método add 637  
   método addFirst 637  
   método addLast 637  
 Linux 13-14, 34-35, 334  
 List, interface 148, 630, 635-636  
   método add 633, 635  
   método addAll 635  
   método clear 635-636  
   método get 633  
   método listIterator 635  
   método size 633, 635-636  
   método subList 635-636  
   método toArray 635-636  
 list, método da classe AssetManager  
   169-170  
 list, método de File 646-647  
 lista 623-624  
 lista de inicializadores 456

lista de itens rolante 106, 265  
 lista de parâmetros 359, 368  
 lista separada por vírgulas  
   de argumentos 338  
   de parâmetros 426  
 lista suspensa 622  
 ListActivity, classe 106-107, 119  
   interface gráfica do usuário  
   personalizada 106-107  
   método getListView 123  
   método setListAdapter 123  
 listagem de código-fonte 2  
 ListFragment, classe 265, 266, 279  
   ListView incorporado 280-281  
   método getListView 281-282  
   método setEmptyText 281-282  
   método setListAdapter 282-283  
 ListIterator, interface 631  
   método hasNext 635-636  
   método previous 635-636  
   método set 635-636  
 listIterator, método da interface  
   List 635  
 ListPreference, classe 144-145  
 ListView, classe 106-107, 279  
   formato de um item da lista 117  
   método setChoiceMode 281-282  
 ListView, vinculação de dados de  
   106-107  
 literal de ponto flutuante 370  
   double por padrão 370  
 load, método da classe SoundPool  
   195-196  
 loadAnimation, método da classe  
   AnimationUtils 146-147, 168-169  
 locais na memória 342  
 Localization Checklist 65-66  
 lockCanvas, método da classe  
   SurfaceHolder 209  
 Log, classe 148, 170  
   método e 170  
 log, método de Math 424  
 logaritmo 424  
 logaritmo natural 424  
 logcat, ferramenta 148  
 LogCat, guia da perspectiva DDMS  
   148  
 Long, classe 630  
 long, palavra-chave 666  
 long, promoções de 430-431  
 loop 386  
   condição de continuação 380  
   contador 397-398  
   corpo 402-403  
   infinito 383  
   instrução 380  
 loop de jogo 186-187, 197-198, 208  
 loop infinito 383, 399-400  
 Lottery Number Picker, exercício do  
   aplicativo 180

## M

Mac OS X 13-14, 34-35, 334  
 main, método 339, 357  
 makeText, método da classe Toast  
   164-165  
 Map, interface 630, 641-642  
   containsKey, método 643-644  
   get, método 643-644  
   isEmpty, método 644-645  
   keySet, método 122  
   put, método 643-644  
   size, método 644-645  
 mapeamento de muitos para um 641  
 mapeamento de um para um 641  
 marcadores de régua (Android  
   Developer Tools) 56  
 marketing viral 323-325  
 Marketwire 326  
 mashup 6  
 Master/Detail Flow, template 44  
 match\_parent, valor da propriedade  
   Layout height 112  
 match\_parent, valor da propriedade  
   Layout width 112  
 Math, classe 402-403, 423  
   abs, método 424  
   ceil, método 424  
   cos, método 424  
   E, constante 424  
   exp, método 424  
   floor, método 424  
   log, método 424  
   max, método 424  
   min 424  
   PI, constante 424, 444-445  
   pow, método 402-403, 423, 424,  
     444-445  
   random, método 432-433  
   sqrt, método 423, 424, 429-430  
   tan, método 424  
 Math.PI, constante 353  
 max, método de Collections 638  
 max, método de Math 424  
 Max, propriedade de uma SeekBar 81  
 Max Length, propriedade de um  
   EditText 80-81  
 máximo divisor comum (MDC) 545  
 média 345  
 média 345, 383, 386  
 média aritmética 345  
 MediaStore, classe 219-220  
 MediaStore.Images.Media, classe 219  
   insertImage, método 219-220  
 melhores práticas x-xi  
 membro de classe não estático 516  
 membro de classe private estático  
   516  
 menor de vários valores inteiros  
   419-420

mensagem 367  
 Menu, classe **143**, 161, 233  
 menu, pasta de um projeto Android **48, 146**  
 menu de opções 19, 22, 138, 140, 217  
 MenuInflater, classe **162**, 233, 284  
   inflate, método **162-163**  
 método **16-17, 334**  
   assinatura **442-443**  
   estático 402-403  
   lista de parâmetros **359**  
   parâmetro **359**, 361  
   tipo de retorno 364  
   variável local **362**  
 método abstrato **552**, 553, 557, 616  
 método chamador **357**, 364  
 método de classe **423**  
 método de comparação natural **638**  
 método de instância (não estático) 516  
 método de predicado 522  
 método exponencial 424  
 métodos de acesso a pacote 520  
 métodos de callback 264-265  
 métodos de ciclo de vida 186-187  
 métodos de ciclo de vida de um aplicativo 85  
 métodos de tratamento de eventos de janela 624-625  
 métodos de visualização de intervalo **635-636**  
 métodos implicitamente final 569  
 microblogs 323-325  
 Microsoft Windows 406-407  
 Miles-Per-Gallon Calculator, exercício do aplicativo 96-97  
 min, método de Collections 638  
 min, método de Math 424  
 MODE\_PRIVATE, constante **122**  
 MODE\_WORLD\_READABLE, constante **122**  
 MODE\_WORLD\_WRITABLE, constante **122**  
 modelo de delegação de evento **615**  
 modelo de retomada de tratamento de exceção **593**  
 modelo de terminação de tratamento de exceção **593**  
 Modificação do sistema de contas a pagar, exercício 584-585  
 Modificação do sistema de pagamento, exercício 584-585  
 modificador de acesso **357**, 363  
   private **363**, 499-500  
   protected 499-500, **526**  
   public **357**, 499-500  
 modificador de acesso protected 499-500, 526  
 modo de escala 246  
 modo de exibição 71-72, **635-636**  
 modo imersivo 22, 216, **219**, 239

modo retrato 195-196  
 modularizando um programa com métodos 423  
 módulos em Java 422  
 monetização de aplicativos 309, 316  
 monitor **652-653**  
 MotionEvent, classe **186**, 207, **219**, 242  
   getActionIndex, método **242-243**  
   getActionMasked, método **242**  
   getPointerCount, método **243**  
   getX, método 243-244  
   getY, método 243-244  
 mouse 608  
 MouseAdapter, classe 624-625  
   mousePressed, método 662-663  
 MouseEvent, classe **617-618**  
 MouseListener, interface **617-618**  
 MouseMotionListener, interface **618-619**, 624-625  
 mousePressed, método da classe MouseAdapter 662-663  
 moveTo, método da classe Path **242**  
 moveToFirst, método da classe Cursor **296-297**  
 Movie Collection, exercício do aplicativo 305-306  
 Movie Trivia Quiz, exercício do aplicativo 180  
 MP3, player 5  
 mudando a escala (números aleatórios) 432-433  
 mudar de diretório 335  
 multimídia x-xi  
 multiplicação, \* **343**, 344  
 MultiSelectListPreference, classe **144-145**  
 multithread **647-648**  
 multitouch 241-242  
 música em fluxo de áudio 195

## N

negação lógica, ! **413-414**  
 new, palavra-chave **340**, **358**, 453, 455  
 New Android Application, caixa de diálogo **40-41**  
 new Scanner(System.in), expressão 340  
 newCachedThreadPool, método da classe Executors **650-651**  
 next, método  
   de Iterator **633**  
   de Scanner **361**  
 nextDouble, método da classe Scanner **373**  
 nextInt, método da classe Random **432-433**  
 nextLine, método da classe Scanner **360**

Nimbus, aparência e comportamento **608**

swing.properties 608  
 nome de classe totalmente qualificado 362  
 nome de classe totalmente qualificado 362  
 nome de menu xix  
 nome de pacote 362  
 nome de um array **453**  
 nome de uma variável **342**  
 nome de versão 311  
 notação algébrica 344  
 notify, método de Object 547-548  
 notifyAll, método de Object 547  
 notifyDataSetChanged, método **126**  
 notifyDataSetChanged, método da classe ArrayAdapter **126**  
 null, palavra-chave 365, **366**, 454  
 NumberFormat, classe **73-74**, 84  
   format, método **87-88**  
 numerando especificadores de formato 149  
 número da posição 452  
 número de ponto flutuante **369**, 387-390  
   divisão 390-391  
   precisão dupla **370**  
   precisão simples **370**  
   tipo primitivo double **369**  
   tipo primitivo float **369**  
 número de ponto flutuante de precisão dupla **370**  
 número de ponto flutuante de precisão simples **370**  
 número primo 493  
 número pseudoaleatório **432**, 433  
 número real 340, 387-388  
 números aleatórios 433-434  
   deslocar um intervalo **432-433**  
   diferença entre valores 433-434  
   escala **432-433**  
   fator de escala **432-433**, 433-434  
   geração 462-463  
   número pseudoaleatório **432-433**  
   processamento 431-432  
   semear **432-433**  
   valor de deslocamento **432**, 433  
   valor semente **433-434**

## O

Object, classe 524, **527**  
 clone, método 546-547  
 equals, método 546-547  
 finalize, método 546-547  
 getClass, método 547-548, **568**  
 hashCode, método 547-548  
 notify, método 547-548



notifyAll, método 547-548  
 toString, método 529, 547  
 wait, método 547-548  
 ObjectInput, interface **647-648**  
   readObject, método **647-648**  
 ObjectInputStream, classe **645**, 647  
 ObjectOutput, interface **647-648**  
   writeObject, método **647-648**  
 ObjectOutputStream, classe **645**, 647  
 objeto (ou instância) 18, 36  
 objeto 16  
 objeto de uma classe derivada 549  
 objeto desserializado **647-648**  
 objeto evento 615-616  
 objeto imutável 517  
 objeto serializado **646-647**  
 obscurecer 311  
 obter um valor com *get* **366**  
 ocultação de dados **363**  
 ocultação de informações **18**, 363  
 ocultar o teclado virtual 123  
 onActivityCreated, método da classe  
   Fragment 190-191  
 onAttach, método da classe Fragment  
   **218**, 249, 280-281, 287-288, 292  
 onCreate, método da classe Activity  
   **71-72**, 185  
 onCreate, método da classe Fragment  
   **144-145**, 175-176  
 onCreate, método da classe  
   SQLiteOpenHelper **302-303**  
 onCreateDialog, método da classe  
   DialogFragment **175**  
 onCreateOptionsMenu, método da  
   classe Activity **143-144**, 161-162  
 onCreateOptionsMenu, método da  
   classe Fragment **233-234**, 294  
 onCreateView, método da classe  
   Fragment **144**, 166, 190  
 onDestroy, método da classe Activity  
   185, **186**  
 onDestroy, método da classe Fragment  
   **186-187**, 191-192  
 onDetach, método da classe Fragment  
   **218**, 249, 280-281, 287-288, 292  
 onDowngrade, método da classe  
   SQLiteOpenHelper **303-304**  
 onDraw, método da classe View **239**  
 OnItemClickListener, interface 281  
 onOptionsItemSelected, método da  
   classe Activity **143-144**, 162-163  
 onOptionsItemSelected, método da  
   classe Fragment **233-234**, 294  
 onPause, método da classe Activity  
   185, **186-187**  
 onPause, método da classe Fragment  
   **186-187**, 191-192, 231-232  
 onPostExecute, método **283**, 284,  
   295

onPostExecute, método da classe  
   AsyncTask **283**, 284, 295  
 onProgressUpdate, método **283**, 295  
 onProgressUpdate, método da classe  
   AsyncTask **283**, 295  
 onResume, método da classe Activity  
   185  
 onResume, método da classe Fragment  
   282-283, 294  
 onSaveInstanceState, método da  
   classe Fragment **264-265**, 294  
 OnSeekBarChangeListener, interface 89  
 onSensorChanged, método **231-232**  
 onSensorChanged, método da interface  
   SensorEventListener **231-232**  
 onSingleTap, método da  
   classe GestureDetector.  
   SimpleGestureListener **240-241**  
 onSizeChanged, método da classe View  
   **195-196**, 237-238  
 onStart, método da classe Activity  
   **161-162**, 185  
 onStart, método da classe Fragment  
   **230-231**  
 onStop, método da classe Activity  
   185  
 onStop, método da classe Fragment  
   **284-285**  
 onTouchEvent, método da classe View  
   **186-187**, 207, **219-220**  
 onTouchEvent, método da classe View  
   241-242  
 onUpgrade, método da classe  
   SQLiteOpenHelper **302-303**  
 onCreateView, método da classe  
   Fragment **280-281**  
 OOAD (análise e projeto orientados  
   a objetos) 18  
 OOP (programação orientada a  
   objetos) **18**, 524  
 Opções para desenvolvedor 10-11  
 Open Handset Alliance 7  
 openPR 326  
 operação em massa **631**  
 operações concorrentes 647-648  
 operações de execução longa 265-266  
 operações paralelas 647-648  
 operador **341**  
 operador binário **341**, 343, 413-414  
 operador condicional, ?: 381  
 operador de adição e atribuição  
   composto, += **394-395**  
 operador de atribuição, = **341**, 349  
 operador de câmera virtual **9-10**  
 operador de comparação 581  
 operador de complemento lógico, !  
   **413-414**  
 operador de decremento, -- **394-395**

operador de decremento pós-fixado  
   **395**  
 operador de decremento prefixado **395**  
 operador de divisão e atribuição  
   composta, /= 395-396  
 operador de exponenciação 402-403  
 operador de incremento pós-fixado  
   **395**  
 operador de incremento prefixado **395**  
 operador de multiplicação e  
   atribuição composto, \*= 395-396  
 operador de resto e atribuição  
   composto, %= 395-396  
 operador de subtração e atribuição  
   composto, -= 395-396  
 operador resto, % **343**, 344  
 operador ternário **381**  
 operador unário **390-391**, 413-414  
   conversão **390-391**  
 operadores  
   ^, OU exclusivo lógico booleano  
     410-411, **412-413**  
   --, pré-decremento/  
     pós-decremento **394-395**  
   --, decremento prefixado/  
     decremento pós-fixado 395-396  
   !, NÃO lógico 410-411, **413-414**  
   ? :, operador condicional ternário  
     381  
   \*=, operador de multiplicação e  
     atribuição 395-396  
   / =, operador de divisão e  
     atribuição 395-396  
   &, E lógico booleano 410, **412**  
   &&, E condicional **410-411**, 411  
   % =, operador de resto e atribuição  
     395-396  
   ++, incremento prefixado/  
     incremento pós-fixado 395-396  
   ++, pré-incremento/  
     pós-incremento 394-395  
   + =, operador de adição e atribuição  
     **394-395**  
   = **341**, 349  
   -=, operador de subtração e  
     atribuição 395-396  
   |, OU inclusivo lógico booleano  
     410-411, **412-413**  
   ||, OU condicional 410, **411**  
   aritméticos **343**  
   atribuição composta 394-395  
   binários **341**, 343  
   complemento lógico, ! **413-414**  
   conversão **390-391**  
   decremento pós-fixado **395-396**  
   decremento prefixado **395-396**  
   E condicional, && **410**, 412  
   E lógico booleano, & **410**, **412**  
   incremento, ++ 395-396

incremento e decremento 395-396  
 incremento pós-fixado **395-396**  
 incremento prefixado **395-396**  
 multiplicação, \* **343**  
 negação lógica, ! **413-414**  
 operador condicional, ?: **381**  
 operador de decremento, -- 394-396  
 operadores lógicos **410-411**, 413-414  
 OU condicional, || 410, **411**, 412  
 OU exclusivo lógico booleano, ^ 410-411, **412-413**  
 OU inclusivo lógico booleano, | **412-413**  
 resto, % **343**, 344  
 subtração, - 344  
 operadores aritméticos 343  
 operadores aritméticos de atribuição compostos **394-395**  
 operadores de atribuição **394-395**  
 operadores de atribuição compostos **394-395**  
 operadores de busca (Twitter) 99  
 operadores de igualdade **346**  
 operadores de incremento e decremento 395-396  
 operadores lógicos **410-411**, 413-414  
 operadores relacionais **346**  
 operando **341**, 390-391  
 ordem 379  
 ordem classificada 641-642  
 ordem crescente 485-486  
 ordem de rotinas de tratamento de exceção 606  
 Ordem dos blocos catch, exercício 606  
 ordem na qual as ações devem ser executadas **378**  
 ordenação com diferenciação de maiúsculas e minúsculas 123  
 ordenação sem diferenciação de maiúsculas e minúsculas 123  
 orientação paisagem 59, 92-93  
 orientação retrato 59, 74-75, 92-93  
 Orientation, propriedade de um GridLayout 76-77  
 origem de evento **613-614**, 615-616  
 OU condicional, || 410, **411**  
 tabela-verdade 412-413  
 OU exclusivo lógico booleano, ^ 410-411, **412-413**  
 tabela-verdade 412-413  
 OU inclusivo lógico booleano, | **412-413**  
 Outline, janela 77-78, 111  
 Outline, janela no Eclipse 70-73  
 OutputStream, classe 647-648  
 @Override, anotação **530-531**

**P**

Package Explorer, janela 185, 263  
 pacote 12, **339**, 422, 430, 518  
 pacote de componentes de interface gráfica do usuário Swing 431-432  
 pacote de entrada/saída 431-432  
 pacote padrão **362**  
 pacotes  
   android.app 13, **71**, 85, 119, 143  
   android.content.res 13, 146, **160**, **168**  
   android.content 13, **106**, **119**, 219  
   android.database.sqlite 13, **265**  
   android.database 13, **265-266**  
   android.graphics.drawable 13, 173  
   android.graphics 13, 187, 219  
   android.hardware 13  
   android.media 13, 186-187  
   android.net 13, **119**  
   android.os 13, 85, 146-147  
   android.preference 13, **143-144**  
   android.provider 13  
   android.text 13, **73-74**, 85  
   android.util 13, 148, 194  
   android.view.animation 146-147  
   android.view.inputmethod **120**  
   android.view 13, **120**, 143-144, 186-187, 219-220  
   android.widget 13, **72-73**, 85, 120, 146-147  
   java.awt.event 431, **615**, 624  
   java.io 13, 173, 431, **645**  
   java.lang **340**, 423, 431-432, 527, 546-547, 648-649  
   java.math 370  
   java.text 13, **73-74**, 84  
   java.util.concurrent 659, 653  
   java.util 13, 148, **339**, 431, 488  
   javax.swing.event **615**, 617, 624  
   javax.swing 431-432, 618-619  
   pacote padrão **362**  
 Padding, propriedade de uma visualização 81-82  
 pagamento 319  
 página da Deitel no Facebook 324  
 Paint, classe **187-188**  
   estilos **237-238**  
   forma preenchida com uma borda **237-238**  
   forma preenchida sem borda **237**  
   linha **237-238**  
   setAntiAlias, método **236-237**  
   setStrokeCap, método 237, **252**  
   setStrokeWidth, método 237-238  
   setStyle, método **236-237**  
 palavra reservada 380  
   false 380

  null 365, **366**  
   true 380  
 palavras-chave  
   abstract **552-553**  
   boolean **381**  
   break **407-408**  
   case **407-408**  
   catch **592**  
   char **340**  
   class **333**, 357  
   continue **410-411**  
   default **407-408**  
   do 380, **402-403**  
   double **340**, **369**  
   else 380  
   enum **437-438**  
   extends **527**, 537-538  
   false **381**  
   final 409-410, **424**, 457  
   finally **592**  
   float **340**, **370**  
   for 380, **398-399**  
   if 380  
   implements **570**  
   import **339**  
   instanceof **566-567**  
   int **340**  
   interface **570**  
   new 340, **358**, 453, 455  
   null **366**, 454-455  
   private **363**, 499-500  
   public **333**, 356, 357, 363, 426, 499-500  
   return **363**, **364**  
   static 402-403, 423  
   super **527**  
   switch 380  
   synchronized **653-654**  
   this **500**, 516  
   throw **600-601**  
   true **381**  
   try **591**  
   void **334**, 357  
   while 380, **402-403**  
 paradas de tabulação 337  
 parâmetro **359**, 361  
 parâmetro de exceção **592**  
 parênteses **334**, 344  
   aninhados **344**  
 parênteses aninhados **344**  
 pares chave/valor associados a um aplicativo 106-107  
 pares chave/valor persistentes 119  
 parse, método da classe Uri **128**  
 passagem por referência **470-471**  
 passagem por valor 468, **470**  
 passar um array para um método 468-469

passar um elemento de array para um método 468-469

pastas

assets **144-145**

res/drawable-mdpi 268-269

Path, classe **219-220**

método moveTo **242-243**

método quadTo **243-244**

método reset **242-243**

Payable, declaração de interface 573-574

pedido fraudulento 319

percorrer um array **478**

Photo Sphere 10-11

pilha 212, **429-430**

  pilha de chamada de métodos **429-430**

  pilha de execução de programa **429-430**

  estouro da pilha **429-430**

  pilha de chamada de métodos **429**

  pilha de execução de programa **429**

  pilha de retrocesso **264-265**, 276, 277, 279

  pop 276

  push **276**

“pinçar” cada algarismo 354

pirataria 312-313

pixels independentes de densidade (dp) **54-55**

pixels independentes de escala (sp) **54-55**

pixels independentes de escala 151

plataformas de aplicativo

  Amazon Kindle 323-324

  Android 323-324

  BlackBerry 323-324

  iPhone 323-324

  Windows Mobile 323-324

play, método da classe SoundPool **199-200**

Play Store, aplicativo 321-322

Plugin ADT para Eclipse 312-313

Plugin Android Development Tools (ADT) 13-14

polimorfismo **547-548**

polinômio 345, 346

polinômio do segundo grau 345, 346

política de licenciamento 311

ponteiro (para eventos de toque) 241-242

ponto de inserção 487-488

ponto de lançamento **589**

ponto e vírgula (;) **334**, 340, 349

Ponto flutuante IEEE 754 666

pool de threads **659-660**

popBackStack, método da classe FragmentManager **276**

pós-decremento **395-396**

pós-incremento **395-396**

postDelayed, método da classe

  Handler **146-147**, 175

potência (expoente) 424

potência de 2 maior que 101 383

pow, método da classe Math **402-403**, 423, 424, 444-445

PR Leap 326

precedência **344**, 349

  operadores aritméticos 344

  tabela 344, 390-391, 664

precedência de operador 344

  regras **344**

  tabela de precedência de operador 390-391, 664

precificação de seu aplicativo 314

precisão

  formato de um número de ponto flutuante 391-392

precisão de um número de ponto flutuante formatado **372**

precisão de um valor de ponto flutuante **369**

preço 315

preço médio de aplicativo pago 315

pré-decremento **395-396**

Preference, classe **144-145**

PreferenceFragment, classe **143**, 175

  método

    addPreferencesFromResource 175

PreferenceManager, classe **144**, 160

  método setDefaultValues 160, **160**

preferências padrão 160-161

pré-incrementando e

  pós-incrementando 396-397

pré-incremento **395-396**

preparado para o futuro 32

*Preparing for Release* 309

pressionamento longo 102-103

previous, método de ListIterator **635-636**

primo 662-663

principal em um cálculo de juros 400-401

princípio do privilégio mínimo **518**

princípios para construção de

  programas 414

print de System.out, método 336

printBitmap, método da classe

  PrintHelper **246**

printf, método de System.out **338**

PrintHelper, classe 246

  método printBitmap **246**

PrintHelper.SCALE\_MODE\_FILL 246

PrintHelper.SCALE\_MODE\_FIT 246

println de System.out, método 336

printStackTrace, método da classe

  Throwable **603-604**

private, modificador de acesso **363**, 499-500

PRLog 326

probabilidade 432-433

probabilidade igual 432-433

problema da média da classe 383, 384, 387-389

problema da média geral da classe 387

problema dos resultados do exame 392-393

procedimento **423**

procedimento para resolver um problema 378

processador de pagamentos 315

processamento de arquivos 645-646

processamento polimórfico de

  exceções relacionadas 596-597

processar objetos Invoice e Employee de forma polimórfica 579

processo de projeto **18**

programa de teste da interface

  Payable processando objetos Invoice e Employee de forma polimórfica 579

programa gerenciador de tela 548-549

programa tolerante à falha **461-462**

programação concorrente **648-649**

programação estruturada **379**

programação orientada a objetos (OOP) **18**, 524

programar no específico 547-548

programar no geral 547-548, 583

Progress, propriedade de uma

  SeekBar 81-82

ProGuard **311**

projeto, adicionar uma classe 189

projeto **40-41**

projeto de interface gráfica do usuário 31

projeto visual de interface gráfica do usuário ix

promoção **390-391**

  de argumentos **429-430**

  regras **429-430**

promoção de argumentos **429-430**

promoções de tipos primitivos 430

prompt **340**

Properties, janela 50-55, 56

protegendo código com um bloqueio **653-654**

provedor de pagamento móvel 317

  Boku 318

  PayPal Mobile Libraries 318

  Samsung In-App Purchase 318

  Zong 318

pseudocódigo **379**, 381, 384, 392

  algoritmo 387-388

**public**

abstract, método 570  
 classe 333  
 dados final estáticos 570  
 interface **496**  
 membro de uma subclasse 527  
 membros de classe estáticos 514  
 método 497, 499-500  
 método estático 516  
 modificador de acesso 356, 357, 363, 426, 499-500  
 palavra-chave **333**, 363  
 serviço **496**

publicar dados em um dispositivo Android 13

publicar uma nova versão de um aplicativo 322-323

put, método da interface Map **643**

putExtra, método da classe Intent **131**

putLong, método da classe Bundle **276**

putString, método da classe SharedPreferences.Editor **126**

**Q**

quadTo, método da classe Path **243**

qualidade sonora 195

qualificado para a coleta de lixo 518

query, método da classe

SQLiteDatabase **300-301**

Queue, interface **630**, **640-641**

quilometragem obtida por

automóveis 418

**R**

R, classe **87**

R.drawable, classe **87**

R.id, classe **87-88**

R.layout, classe **87-88**

R.layout.activity\_main, constante **87-88**, 121

R.string, classe **87-88**

radianos 424

raio de um círculo 545

raiz quadrada 424

Random, classe 431-432, **432-433**

nextInt, método **432-433**

setSeed, método 433-434

random, método da classe Math 432

range, método da classe EnumSet **514**

rastro da pilha **588**

raw, pasta de um projeto Android **48**, **146**

readObject, método de ObjectInput **647-648**

realce de código xii-xiii, 2

realização na UML **572-573**

realizar uma tarefa 357

recebendo eventos **613-614**

receptor de evento 581, 615, 624

classe adaptadora 624-625

interface **612**, 613, 616-618, 624

receptor registrado 617-618

reconhecimento de voz ix-xi

Rectangle, Classe (exercício) 521

recuo 382

recursão infinita 546-547

recurso 320

recurso de aplicativo 13

recurso de dimensão 112

recurso de estilo 270-273

recursos 62-63

adaptados para o local 61-62

android-developers.blogspot.com/ 33-34

androiddevweekly.com/ 33-34

answers.oreilly.com/

topic/862-ten-tips-for-android-application-development/ 33-34

code.google.com/p/

apps-for-android/ 33-34

convenções de atribuição de

nomes de recurso alternativas 61

cyrimottier.com/ 33-34

developer.motorola.com/ 33-34

developer.sprint.com/site/

global/develop/mobile\_

platforms/android/android.jsp

33-34

graphics-geek.blogspot.com/ 33

Localization Checklist 65-66

padrão 61-62

stackoverflow.com/tags/android/

topusers 33-34

style **264-265**

www.brighthub.com/mobile/

google-android.aspx 33-34

www.curious-creature.org/

category/android/ 33-34

www.htcdev.com/ 33-34

recursos adaptados ao local 61-62

recursos de aplicativo (developer.

android.com/guide/topics/

resources/index.html) **52**

recursos de relações públicas na Internet

ClickPress 326

i-Newswire 326

Marketwire 326

Mobility PR 326

openPR 326

PR Leap 326

Press Release Writing 326

PRLog 326

PRWeb 326

recursos padrão 61-62

red, método da classe Color **249**

redação de press release 326

rede de anúncios móveis 316, 326

AdMob 316, 327

Flurry 327

InMobi 327

JumpTap 327

Medialets 327

mMedia 327

Nexage 327

Smaato 327

Tapjoy 327

redes sociais 323-325

redesenhar uma View 239-240

redimensionamento dinâmico **452**

referência **366**

reflexão 568

registerListener, método da classe

SensorManager **230-231**

registerOnSharedPrefer-

enceChangeListener, método da

classe SharedPreferences **160-161**

registrando a rotina de tratamento de

evento **612-613**

registrando exceções **148**, 170

registro como desenvolvedor 318

registro de ativação **429-430**

registro de evento 613-614

regras de precedência de operador **344**

reinventando a roda 339, 485-486

reivindicar memória 518-519

relação *é um* **524**

relação *tem um*, **524**

relações públicas 325-326

Relançando Exceções, exercício 606

relançar uma exceção **600-601**, 606

RelativeLayout **48**

release, método da classe SoundPool **205-206**

remove, método da classe

ArrayList<T> 488, **490**

remove, método da interface Iterator **633**

remover aplicativos do mercado 322-323

renda com anúncios 316

renderização e monitoramento de texto 13

repetição

controlada por contador 384, 389

controlada por sentinela 387-389

repetição controlada por contador

**383**, 384, 389-390, 397-399

repetição controlada por sentinela

**387-388**, 388-389

repetição indefinida **387-388**

repetição termina 383

replace, método da classe  
 FragmentTransaction **276**  
 reprodução de áudio ix-x  
 requisito de capturar ou declarar **595**  
 requisitos **18**  
 requisitos do sistema operacional xix  
 res, pasta de um projeto Android **46-47, 52**  
 res/de, pasta um projeto Android **185, 189-190**  
 res/drawable-mdpi, pasta 268-269  
 res/raw, pastas **185, 189-190**  
 reset, método da classe Path **242**  
 Resource Chooser, caixa de diálogo 52-55  
 Resources, classe **160-161, 168-169**  
   getConfiguraton, método **160**  
   getString, método **168-169**  
 respostas a um levantamento 460  
 resto 343  
 resumindo respostas a um levantamento 460  
 retângulo 521  
 retorno de carro 337  
 retorno visual 621  
 return, palavra-chave **364**  
 reutilização **16-17, 16-17, 339**  
 reutilização de código 524  
 reutilização de software **423, 520, 524**  
 reverse, método de Collections 638  
 reverseOrder, método de Collections **639**  
 RGB, valores **81-82**  
 RGB 24-25  
 Road Sign Quiz, exercício do aplicativo 180  
 robusto 341  
 rolar 623  
 rotate, animação de para um View **157-158**  
 rotina de tratamento de evento 581, **609**  
   retornando false **240-241**  
 rotina de tratamento de exceção **592**  
 rotina de tratamento de exceção padrão 603-604  
 rótulo 309  
 rótulo de botão **618-619**  
 rótulo em um switch **407-408**  
 run, método da interface Runnable **648-649**  
 Runnable, interface 146-147, 205-206  
 Runnable, interface 581, **648-649**  
   run, método **648-649**  
 runOnUiThread, método da classe Activity **205-206**  
 RuntimeException, classe **595**

## S

saída 334  
 saída formatada  
   alinhamento à direita **401-402**  
   alinhar à esquerda **401-402**  
   especificador de formato %f **371**  
   flag 0 **459, 498**  
   flag de formatação - (sinal de subtração) **401-402**  
   flag de formatação , (vírgula) **402**  
   flag de formatação sinal de subtração - **401-402**  
   flag de formatação vírgula (,) **402**  
   largura de campo **401-402**  
   números de ponto flutuante **371**  
   precisão **372**  
   separador de agrupamento **402**  
 SalariedEmployee, classe concreta  
   estende a classe abstrata Employee 558-559  
 SalariedEmployee, classe  
   que implementa o método  
   getPaymentAmount da interface Payable 578  
 SavingsAccount, Classe (exercício) 521  
 Scale Type, propriedade de um ImageView 154  
 SCALE\_MODE\_FILL 246  
 SCALE\_MODE\_FIT 246  
 Scanner, classe 339, **340**  
   hasNext, método **407-408**  
   next, método **361**  
   nextDouble, método **373**  
   nextLine, método **360**  
 Scrapbooking, exercício do aplicativo 67  
 ScrollView, classe **269-270**  
 SDK Android 2.x vii, x-xi  
 SDK Android xix, xxii, **2, 13-14, 34-35**  
 SeekBar  
   Max, propriedade 81-82  
   Progress, propriedade 81-82  
 SeekBar, classe 70-71, **72-73, 85**  
 SeekBar.OnSeekBarChangeListener, interface 73-74, 85, 249-250  
 segundo plano  
   atividade enviada para o 191-192  
 segurança quanto ao tipo em tempo de compilação 629  
 selecionador de intenção 104, **107**  
 selecionando um item de um menu 609  
 senha 609  
 seno 424  
 seno trigonométrico 424  
 Sensor, classe **218**

Sensor.TYPE\_ACCELEROMETER, constante 230-231  
 sensor acelerômetro **218, 231-232**  
 sensor de aceleração linear 218  
 sensor de campo magnético 218  
 sensor de gravidade 218  
 sensor de luz 218  
 sensor de orientação 218  
 sensor de pressão 218  
 sensor de proximidade 218  
 sensor de temperatura 218  
 sensor de vetor de rotação 218  
 sensor giroscópio 218  
 Sensor Simulator 15  
 SENSOR\_DELAY\_NORMAL, constante da classe SensorManager 230-231  
 sensores  
   aceleração linear 218  
   acelerômetro **218, 231-232**  
   campo magnético 218  
   giroscópio 218  
   gravidade 218  
   luz 218  
   orientação 218  
   pressão 218  
   proximidade 218  
   temperatura 218  
   vetor de rotação 218  
 SensorEvent, classe **232-233**  
 SensorEventListener, interface 231  
 SensorEventListener, receptor **231**  
 SensorManager, classe 230-231  
   getDefaultSensor, método **230**  
   registerListener, método **230**  
   unregisterListener, método **231**  
 SensorManager.SENSOR\_DELAY\_NORMAL, constante 230-231  
 separador de agrupamento (saída formatada) **402-403**  
 separador ponto (.) **358, 402-403, 423, 514-515**  
 sequência de escape **337, 340**  
   \\, barra invertida 337  
   \\", aspa dupla 337  
   \\t, tabulação horizontal 337  
   nova linha, \\n 337, 340  
 sequência de escape com nova linha, \\n 337, 340  
 Serializable, interface 581  
 serialização de objetos ix-x, **646-647**  
 série de Fibonacci 494  
 série infinita 419-420  
 serviço de licenciamento **311**  
 serviço de uma classe 499-500  
 serviços de jogo do Google Play ix-x  
 serviços dos sistemas operacionais 13  
 serviços web **6**  
   Amazon eCommerce 6  
   eBay 6



- Facebook 6
- Flickr 6
- Foursquare 6
- Google Maps 6
- Groupon 6
- Instagram 6
- Last.fm 6
- LinkedIn 6
- Microsoft Bing 6
- Netflix 6
- PayPal 6
- Salesforce.com 6
- Skype 6
- Twitter 6
- WeatherBug 6
- Wikipedia 6
- Yahoo Search 6
- YouTube 6
- Zillow 6
- Set, interface **148, 630, 640, 641**
- set, método 366, 501-502
- set, método da interface ListIterator **635-636**
- set em uma animação **157-158**
- seta de rolagem **623-624**
- setAntiAlias, método da classe Paint **236-237**
- setArguments, método da classe Fragment **276**
- setBackgroundColor, método **249**
- setBackgroundColor, método da classe View **249-250**
- setChoiceMode, método da classe ListView **281-282**
- setContentView, método da classe Activity **87-88**
- setDefaultValues, método da classe PreferenceManager 160, **160**
- setEditable, método da classe JEditTextComponent **612-613**
- setEmptyText, método da classe ListFragment **281-282**
- setImageBitmap, método da classe View **253-254**
- setImageDrawable, método da classe InputStream **173**
- setLayout, método da classe Container **611**
- setLayout, método da classe Container **611**
- setListAdapter, método da classe ListActivity **123**
- setListAdapter, método da classe ListFragment **282-283**
- setMaximumRowCount, método da classe JComboBox **623-624**
- setRepeatCount, método da classe Animation **148, 168-169**
- setRequestedOrientation, método da classe Activity **160-161**
- setRetainInstance, método da classe Fragment **281-282**
- setRolloverIcon, método da classe AbstractButton **621**
- setSeed, método da classe Random 433-434
- setStrokeCap, método da classe Paint 237-238, **252-253**
- setStrokeWidth, método da classe Paint 237-238
- setStyle, método da classe Paint **236-237**
- setSystemUiVisibility, método da classe View **240-241**
- Setting hardware emulation options 29-30
- setVolumeControlStream, método da classe Activity **186-187, 190-191**
- shape, elemento **268-269**
- shape, elemento do recurso Drawable **268-269**
- Shape, hierarquia de classes 526
- SharedPreferences, classe **106-107, 119, 120**
  - edit, método **126**
  - getAll, método **122**
  - getString, método **127**
  - getStringSet, método **164-165**
  - registerOnSharedPreferenceChangeListener, método **160**
- SharedPreferences.Editor, classe **106-107, 126**
  - apply, método **126**
  - putString, método **126**
- shell **334**
- Shopping List, exercício do aplicativo 305-306
- Short, classe **630**
- short, tipo primitivo 403-404, 666
  - promoções 430-431
- show, método da classe DialogFragment **175**
- shuffle, método da classe Collections **148, 638, 640**
- shutdown, método da classe ExecutorService **652-653**
- SimpleCursorAdapter, classe **281-282**
- SimpleOnGestureListener, interface 240-241
- simulação 431-432
  - lançamento de moeda 545
- sin, método da classe Math 424
- sincronização **652-653**
- sincronizar **648-649**
- síntese de voz ix-xi
- sistema operacional 7
- site da Deitel na Web ([www.deitel.com](http://www.deitel.com)) xxv
- site de acompanhamento ix-x
- sites de análise de aplicativo
  - Android and Me 325-326
  - Android App Review Source 325
  - Android Police 325-326
  - Android Tapp 325-326
  - AndroidGuys 325-326
  - AndroidLib 325-326
  - AndroidPIT 325-326
  - AndroidZoom 325-326
  - Androinica 325-326
  - AppBrain 325-326
  - Appolicious 325-326
  - Appstorm 325-326
  - Best Android Apps Review 325
  - Phandroid 325-326
- sites de análise de aplicativo em vídeo
  - Android Video Review 325-326
  - Appolicious 325-326
  - Crazy Mike's Apps 325-326
  - Daily App Show 325-326
  - Life of Android 325-326
- sites de mídia social 323-324
- size, método
  - da classe ArrayList<T> **490**
  - da interface List **633, 635-636**
  - da interface Map **644-645**
- sleep, método da classe Thread **650-651**
- SMS 104-105
- sobrecarregar um método **440-441**
- sobrescrever um método de superclasse **526, 529-530**
- Software Development Kit Java SE 7 xix
- software frágil **542-543**
- Solitaire Card Game, exercício do aplicativo 136
- sombreamento de sintaxe xii-xiii
- sombrear um campo **438-439**
- sons 185
- sort, método
  - da classe Arrays **485-486**
  - da classe Collections **123, 638**
- SortedMap, interface **641-642**
- SortedSet, interface **641-642**
- SoundPool, classe **186-187, 195**
  - método load **195-196**
  - método play **199-200**
  - método release **205-206**
- sp (pixels independentes de escala) 54-55
- Sports Trivia Quiz, exercício do aplicativo 180
- SQL (Structured Query Language) 265-266
- SQLite 13, 260, 265-266

SQLiteDatabase, classe **265-266**  
   delete, método **301-302**  
   execSQL, método **303-304**  
   insert, método **299-300**  
   query, método **300-301**  
   update, método **300-301**  
 SQLiteOpenHelper, classe **265-266**,  
 298-299, 302-303  
   método getWritableDatabase **298**  
   onCreate, método **302-303**  
   onDowngrade, método **303-304**  
   onUpgrade, método **302-303**  
 sqrt, método da classe Math 423, 424,  
 429-430  
 stack frame **429-430**  
 stack unwinding **601-602**  
 StackTraceElement, classe **603-604**  
   método getClassName **603-604**  
   método getFileName **603-604**  
   método getLineNumber **603-604**  
   método getMethodName **603-604**  
 startActivity, método da classe  
 Context **107-108**, 128  
 startAnimation, método da classe  
 View **148**  
 static  
   campo (variável de classe) **514**  
   membro de classe 514-515  
   método 357, 402-403  
   palavra-chave 423  
   variável de classe 516  
 streaming 13  
 String, classe  
   imutável **517**  
   método format **498**  
   toLowerCase **635-636**  
   toUpperCase **635-636**  
 String, recurso de contendo vários  
 especificadores de formato 149  
 String, remover duplicada 640-641  
 String.CASE\_INSENSITIVE\_ORDER,  
 objeto de Comparator<String> 123  
 string **334**  
   de caracteres 334  
   literal **334**  
 String codificada em URL 127  
 string de caracteres **334**  
 string de formato **338**  
 string vazia **614-615**  
 strings, concatenação de **427**, 517  
 strings.xml **52**, 79-80, 111  
 strings de formatação 149  
 Strings em instruções switch 409  
 stroke, elemento de um objeto shape  
**268-269**  
 Structured Query Language (SQL)  
 265-266  
 style, atributo de um componente de  
 interface gráfica do usuário **264-265**

Style, propriedade de um View 270  
 style, recursos **264-265**  
 styles.xml 267-268  
 subclasse 71-72, **524**  
 subclasse concreta 557-558  
 subList de List, método **635-636**  
 sublista **635-636**  
 subscrito (índice) **452**  
 subtração 343  
   operador, - 344  
 super, palavra-chave 527  
   chamar construtor da superclasse  
   539-540  
 superclasse **524**  
   construtor 529-530  
   construtor padrão 529-530  
   direta **524**, 526  
   indireta **524**, 526  
   método sobrescrito em uma  
   subclasse 546-547  
   sintaxe de chamada de construtor  
   **539-540**  
 superclasse abstrata **552-553**  
 superclasse direta 524, 526  
 superclasse indireta 524, 526  
 suporte para hardware 13  
 suporte para orientações retrato e  
 paisagem 112  
 surfaceChanged, método da interface  
 SurfaceHolder.Callback **206**  
 surfaceCreated, método da interface  
 SurfaceHolder.Callback **206**  
 surfaceDestroyed, método da  
 interface SurfaceHolder.Callback  
**206**  
 SurfaceHolder, classe **187-188**, 195  
   addCallback, método **195**  
   lockCanvas, método **209**  
 SurfaceHolder.Callback, interface  
**187-188**, 195, 206  
   surfaceChanged, método **206**  
   surfaceCreated, método **206**  
   surfaceDestroyed, método **206**  
 SurfaceView, classe **187-188**, 195  
   getHolder, método **195**  
 swing.properties, arquivo 608  
 SwingConstants, interface 581  
 SwingWorker, classe **654-655**  
   done, método 654-655, 657  
   execute, método 654-655  
   get, método 654-655  
   método doInBackground 654, 657  
   process, método 654-655  
   publish, método 654-655  
   setProgress, método 654-655  
 synchronized 209  
   instrução **653-654**  
   método **653-654**  
   palavra-chave **653-654**

System, classe  
   arraycopy **485-486**, 487-488  
   exit, método **597-598**  
 System.err (fluxo de erro padrão) **592**  
 System.out  
   print, método **336**, 336  
   printf, método **338**  
   println, método **334**, 336  
 System.out (fluxo de saída padrão) **334**  
 SYSTEM\_UI\_FLAG\_FULLSCREEN 240-241  
 SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION  
 240-241  
 SYSTEM\_UI\_FLAG\_IMMERSIVE 240-241  
 SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN  
 240-241  
 SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_  
 NAVIGATION 240-241  
 SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE 240

## T

tabela 476-477  
 tabela de valores 476-477  
 tabela hash 640-641  
 tabela-verdade **411-412**  
   do operador ^ 412-413  
   do operador ! 413-414  
   do operador && 411-412  
   do operador || 411-412  
 tabela-verdade da negação lógica, ou  
 operador NÃO lógico (!) 413-414  
 TableLayout, célula em um 74-75  
 TableLayout, classe **74-75**  
 tablet 8-9  
 tabulação horizontal 337  
 TalkBack **39-40**, **59**, 112, 114  
   adaptação ao local 64-65  
 tamanho da fonte 187-188  
 tamanho de uma variável **342**  
 tan, método da classe Math 424  
 tangente 424  
 tangente trigonométrica 424  
 Target-Heart-Rate Calculator,  
 exercício do aplicativo 96-97  
 taxa de juros 400-401  
 teclado 5, 338, 608  
 teclado numérico 69  
 teclado numérico virtual 92-93  
 teclado virtual  
   impedir a exibição na inicialização  
   do aplicativo 133  
   impedir a exibição no  
   carregamento do aplicativo 109  
   permanece na tela 74-75  
   tipos 271-272  
 tela multi-touch 5  
 Tema  
   Holo Dark 42-43  
   Holo Light 42-43

tema Holo Light With Dark  
Action Bars 42-43

templates de projeto **44**

- Blank Activity **44**
- Fullscreen Activity **44**
- Master-Detail Application **44**

temporário 390-391

Text, propriedade de um componente **52**

Text Appearance, propriedade de  
TextView 77-78

Text Appearance, propriedade de um  
TextView 77-78

Text Color, propriedade de um  
componente **55**

Text Size, propriedade de um  
componente **54-55**

texto fixo 341

- em uma string de formato **338**

TextView, classe **39-40**, 52, 72-73, 85

- propriedade Label For 80-81

TextView, componente **49**

TextWatcher, interface **73-74**, 85

The Java™ Language Specification  
344

this

- palavra-chave **500**, 516
- para chamar outro construtor da  
mesma classe 505-506
- referência **500**

thread (para animação) 186-187

Thread, classe 86-187, 208

- interrupt, método **650-651**
- sleep, método **650-651**

thread **593**

- de execução **647-648**
- sincronização **652-653**

thread da interface gráfica do usuário  
265-266

thread de despacho de evento (EDT)  
**653-654**

thread principal **652-653**

thread segura 653-654

throw, instrução **498**, **599-600**

throw, palavra-chave **600-601**

throw para lançar exceção **498**, 506

Throwable, classe **594-595**, 603-604

- getMessage, método **603-604**
- getStackTrace, método **603-604**
- printStackTrace, método **603-604**

throws, cláusula **593**

TicTac-Toe (Jogo da Velha), exercício  
522

Time2, classe, melhorando a  
(exercício) 521

Tip Calculator, aplicativo ix, 15

tipo **340**

tipo de dado abstrato (ADT) 497

tipo de referência **366**, 520

tipo de retorno 364

- de um método **357**, 364

tipo de uma variável **342**

tipo primitivo **340**, 366, 396, 430

- byte 403-404
- char 340, 403-404
- double **340**, **369**, **370**, 387-388
- float **340**, **369**, **370**
- int 340, 387-388, 395-396, 403
- nomes são palavras-chave 340
- passado por valor 470-471
- promoções 430-431
- short 403-404

tipos de teclado 271-272

toArray, método de List **635**, 638

Toast, classe **146-147**, 164-165

- método makeText **164-165**

tolerante à falha 341

toLowerCase, método da classe String  
**635-636**

tomando decisões 350

toString, método da classe Object  
529-530, 547-548

total **384**

toUpperCase, método da classe String  
**635-636**

Tower of Hanoi (Torre de Hanói),  
exercício do aplicativo 212

transação financeira 318

transferência de controle **379**

translate, animação de

- android:duration, atributo **158**
- android:fromXDelta, atributo **157**
- android:startOffset, atributo **158**
- android:toXDelta, atributo **157**

translate, animação de para um View  
**157-158**

transparência 81-82, 217

tratamento de evento 71, **609**, 612,  
616

- origem de evento **613-614**

tratar uma exceção 589

TreeMap, classe **641-642**

TreeSet, classe **640-641**, 641-642

trimToSize, método da classe  
ArrayList<T> 488

troca de mensagens de intenção  
107-108

true, palavra reservada 380, **381**

true **346**

truncar 343

truncar a parte fracionária de um  
cálculo **387**

try, bloco **461-462**, 591, 602-603

- termina 593

try, instrução **461-462**, **593**

try, palavra-chave **591**

tweet 324-325

Twitter, busca no 99

- operadores 101

Twitter 6, 104-105, **324-325**

- @deitel 324-325
- hashtag 324-325
- tweet 324-325

Twitter Searches, aplicativo ix

Twitter Searches, exercício do  
aplicativo

- aprimoramentos 135
- com fragmentos 180

TYPE\_ACCELEROMETER, constante da  
classe Sensor 230-231

## U

U.S. State Quiz, exercício do  
aplicativo 180

último a entrar, primeiro a sair (LIFO  
- last-in, first-out) **429-430**

Uniform Resource Identifier (URI)  
**645-646**

Uniform Resource Locator (URL)  
**645-646**

UNIX 334, 406-407

unregisterListener, método da classe  
SensorManager **231-232**

UnsupportedOperationException,  
classe **635-636**

update, método da classe  
SQLiteDatabase **300-301**

URI (Uniform Resource Identifier)  
**645-646**

Uri, classe **119**, **128**

- método parse **128**

URL (Uniform Resource Locator)  
**645-646**

USB debugging 29-30

Use Default Margins, propriedade de  
um GridLayout 76-77

utilitários 31

Utilities Package 431-432

## V

valor absoluto 424

valor com sinal **387-388**

valor de deslocamento (números  
aleatórios) 433-434

valor de deslocamento **432-433**

valor de uma variável **342**

valor dummy **387-388**

valor final 398-399

valor flag **387-388**

valor inicial de variável de controle  
**397-398**

valor inicial padrão **365**

valor padrão **365**

valor semente (números aleatórios)  
432-433, **433-434**



valor sentinela **387-388**, 390-391  
 valores alfa (transparência) 81-82  
 valores para o inteiro mais próximo 447-448  
 values, método de um enum **513**  
 values, pasta de um projeto Android **48, 52**  
 várias declarações de classe em um único arquivo de código-fonte 500  
 variável **338**, 340  
   nome **340, 342**  
   tamanho **342**  
   tipo **342**  
   tipo de referência **366**  
   valor **342**  
 variável constante **409**, 457, 518  
   deve ser inicializada 457  
 variável de ambiente  
   CLASSPATH 336  
   PATH 335  
 variável de ambiente PATH 335  
 variável de classe **424, 514-515**  
 variável de controle **383, 397**, 398  
 variável de instância **18, 362**, 363, 370, 424  
 variável local **362**, 386, 438-440, 501  
 variável não pode ser modificada 518-519  
 vários especificadores de formato 149  
 vazamento de memória 514-515, 597-598  
 vazamento de recurso 514, **597-598**  
 Vector, classe 491  
 verificação de limites **460**  
*Versioning Your Applications* 311  
 versões de Android  
   Android 1.5 (Cupcake) 7  
   Android 1.6 (Donut) 7  
   Android 2.0 a 2.1 (Eclair) 7  
   Android 2.2 (Froyo) 7  
   Android 2.3 (Gingerbread) 7  
   Android 3.0 a 3.2 7  
   Android 4.0 (Ice Cream Sandwich) 7  
   Android 4.1 a 4.3 7  
   Android 4.4 7  
 versões do SDK Android e níveis de API 40-42

vídeo de demonstração de Netbeans 332  
 vídeo ix-xi, 13  
 vídeo viral 324-325  
 videogame 432-433  
 View, animações de **157-158**  
 View, classe **120**, 187-188, **249-250**  
   getSystemUiVisibilty, método **240-241**  
   invalidate, método **238-239**  
   mudanças de tamanho 195-196  
   onDraw, método **239-240**  
   onSizeChanged, método **195**, 237  
   onTouchEvent, método **186**, 207, **219**, 241  
   redesenhar um View 239-240  
   setImageBitmap, método **253-254**  
   setSystemUiVisibility, método **240-241**  
   startAnimation, método **148**  
   subclasse personalizada 192  
 View, qualificar totalmente o nome de classe de uma personalizada em um layout XML 185  
 View, subclasse personalizada de 192  
 View.OnClickListener, interface **120**  
 View.SYSTEM\_UI\_FLAG\_FULLSCREEN 240-241  
 View.SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION 240-241  
 View.SYSTEM\_UI\_FLAG\_IMMERSIVE 240-241  
 View.SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN 240-241  
 View.SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_NAVIGATION 240-241  
 View.SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE 240-241  
 ViewGroup, classe **269-270**  
 Views aninhados **269-270**  
 vinculação de dados 106-107  
 vinculação dinâmica **566**  
 vinculação estática **569**  
 vinculação tardia **566**  
 vinculando seus aplicativos 321-322  
 vincular dados a um ListView 106  
 visualização personalizada 185  
 void, palavra-chave **334**, 357

volume 186-187  
 volume de uma esfera 444-447  
 volume do áudio 186-187

## W

wait, método da classe Object 547  
 Weight, propriedade de um componente 82-83, 114  
 Weight, propriedade de um componente de interface gráfica do usuário 153  
 Welcome, aplicativo ix, 13-14, 15  
 Welcome, guia no Eclipse 40-41  
 while, instrução de repetição 380, **383**, 386, 389-391, 397-398  
 widget 13, 85, 120, 608  
 widgets de bloqueio de tela 10-11  
 Wi-Fi Direct 9-10  
 window gadgets 608  
 Window soft input mode, opção 92-93, 133  
 WindowListener, interface 624-625  
 WindowManager, classe **146**, 161  
 Windows 13-14, 34-35, 406-407  
 Word Scramble Game, exercício do aplicativo 135  
 Word Search, exercício do aplicativo 257-258  
 Workspace Launcher, janela 19  
 wrap\_content, valor do atributo android:layout\_height 80-82  
 wrap\_content, valor do atributo android:layout\_height 80-82  
 wrap\_content, valor do atributo android:layout\_width 80-82  
 wrap\_content, valor do atributo android:layout\_width 80-82  
 writeObject, método da interface ObjectOutputStream **647-648**

## X

xml, pasta de um projeto Android **48, 146**  
 XML, utilitários 13

## Y

YouTube 313-314